Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL FINAL DE GRAU

TÍTOL DEL TFG: Plataforma per la gestió d'equipatges del sector aeroportuari

TITULACIÓ: Grau en Enginyeria Telemàtica

AUTOR: Marc Bernat Saiz

DIRECTOR: Toni Oller Arcas

DATA: 4 de setembre del 2017

Títol: Plataforma per la gestió d'equipatges del sector aeroportuari

Autor: Marc Bernat Saiz

Director: Toni Oller Arcas

Data: 4 de setembre del 2017

Resum

Uns dels problemes actuals en el sector aeroportuari és la pèrdua dels equipatges en els diversos punts de la infraestructura logística dels aeroports. En la pràctica, es perden més de 6 maletes per cada 1000 passatgers. I en total, un conjunt de 23 milions de maletes extraviades cada any. Una situació insostenible de 2.300 milions d'euros en despeses associades a la gestió de reclamacions i el pagament de compensacions econòmiques.

Així doncs, una de les principals debilitats en els sistemes aeroportuaris, és l'ús de mecanismes manuals per la identificació de les maletes. En altres paraules, la revisió de l'equipatge es realitza amb una inspecció visual o amb lectors de codis de barres, ambdós requerint un camp de visió directa. Això provoca demores en la localització de les maletes i la insatisfacció del client en la resolució del cas.

Donat el problema, es proposa la construcció d'una plataforma adaptada a suplir les necessitats actuals utilitzant les últimes tecnologies disponibles. Aquesta està composta per un portal web, una aplicació mòbil, i per últim, un servidor web que permeti la gestió i la consulta de les dades. El portal web donarà suport als administradors del sistema i als operadors dels aeroports, mentre que l'aplicació mòbil, facilitarà el seguiment de l'equipatge per als usuaris finals.

La principal innovació del projecte recau en la utilització de tecnologia RFID. Cada equipatge incorpora un clauer RFID que identifica la maleta amb les dades emmagatzemades en la base de dades del sistema. A diferència dels sistemes anteriors, la lectura es realitza sense contacte visual i permet l'automatització del procés. També, es desenvolupa un equip lector utilitzant una Raspberry Pi en conjunt d'un mòdul RFID amb la finalitat de llegir els clauers com antena receptora dins de la plataforma.

En conclusió, l'objectiu del projecte és oferir una plataforma que permeti la gestió dels equipatges utilitzant clauers RFID per tal de reduir despeses econòmiques, la millora general en la resolució dels casos i evitar la pèrdua de l'equipatge.

Overview

One of the current problems in the airport sector is the baggage losses at the various points of the airport logistics infrastructure. In fact, more than 6 suitcases for each 1,000 passengers are lost. Altogether, a total of 23 million bags are lost each year. This is an unsustainable situation of 2.3 million euros in expenses associated with handling claims and the payment of financial compensations.

Therefore, one of the main weaknesses in airport systems is the usage of manual mechanisms for the baggage identification. In other words, luggage inspection is carried out with visual inspection or using barcode readers, both of which require a direct field of view. This causes delays in the identification of suitcases and customer dissatisfaction in the resolution of the problem.

With this problem, it’s proposed to build a platform adapted to satisfy the current needs using the latest technologies available. It consists in a Web portal, a mobile application, and lastly, a web server that allows the management and query the database. The web portal will support system administrators and airport operators, meanwhile, the mobile application will provide baggage tracking for end-users.

The main innovation of the project consists in the utilization of RFID technology. Each baggage incorporates an RFID tag that identifies the suitcase with the data stored in the system's database. Unlike previous systems, the reading is done without visual contact and allows to automate the process. Also, it is developed a reader device using a Raspberry Pi in conjunction with an RFID module with the purpose of reading the tags as antenna receiver inside the platform.

In conclusion, the project's objective is to offer a platform that allows the management of baggages with RFID keys in order to reduce economic expenses, a general improvement on the resolution of problems and avoid the loss of personal baggages.

ÍNDEX

ÍNDEX DE FIGURES

ÍNDEX DE TAULES

Dedicatòria

Voldria dedicar aquest treball a la meva mare Ester i al meu pare Jordi pels seus esforços continuats en mi i per motivar-me en aconseguir tots els reptes que em proposi.

A la meva parella Denitsa per ajudar-me en tots els aspectes de la vida i tindre-la sempre al meu costat en els moments difícils.

Per últim, agrair el meu tutor Toni per l'elaboració d'aquest treball.

INTRODUCCIÓ

Un dels problemes habituals a l'hora de viatjar amb avió és la possible pèrdua de l'equipatge personal. A causa d'una mala gestió logística, l'origen del problema prové en la implementació d'un sistema totalment dependent dels recursos humans. Això impossibilita l'automatització del seguiment d'una maleta de forma àgil.

Un altre punt feble és la transferència de dades entre el mateix sistema i les etiquetes incloses en les maletes. Necessitant una visió directa amb l'etiquetatge que dificulta la pròpia lectura. Per tant, és un mètode poc pràctic pel nombre d'equipatges que circulen per les infraestructures aeroportuàries.

Així doncs, aquest projecte té com a objectiu: la millora de la traçabilitat i l'eficiència en el sector aeroportuari. Per això, s'utilitzaran les noves tecnologies, com GraphQL i Angular 4, per poder realitzar una millora en aquest àmbit amb la creació d'una plataforma en línia que permeti tenir un control total sobre els diversos models de dades.

En referencia pel que fa al document, està estructurat en diferents capítols amb la finalitat d'exposar i explicar les diverses funcionalitats del projecte, l'arquitectura d'aquest i la implementació entre altres. En conclusió, el projecte s'organitza de la següent forma:

Al primer capítol es contextualitza la problemàtica actual on està realitzat el projecte. Compta amb una exposició de l'estat actual del problema, les necessitats a cobrir per part del projecte, i la definició dels objectius a realitzar.

Al segon capítol es vol informar sobre les especificacions tècniques del projecte. En altres paraules, la presentació de funcionalitats, els rols disponibles i el hardware especialitzat per l'execució del projecte.

En el tercer capítol, s'exposa l'arquitectura del projecte. Fent èmfasis en l'estructura del model de dades.

En el quart capítol, es descriu amb diversos apartats, la forma d'implementar els objectius i funcionalitats en l'àmbit del projecte. En cada apartat es concreten les tecnologies emprades, les definicions, i els exemples/explicacions de cada implementació realitzada. Incloent l'entorn de desenvolupament.

Al cinquè capítol s'explica la planificació seguida durant el desenvolupament del treball.

I per últim, en el sisè capítol s'exposarà les conclusions finals del projecte, futures millores, i un estudi de l'impacte mediambiental produït.

CAPÍTOL 1. CONTEXT

En el present capítol es vol exposar el context en el qual el projecte està realitzat. Des d'un punt de partida, es voldrà estudiar el problema actual, analitzar les necessitats a cobrir, i per últim, els objectius que es volen aconseguir amb la realització del projecte.

1.1. Punt de partida

En 2007, les companyies aèries han intensificat la creació de nous sistemes de rastreig i monitoratge d'equipatges [1]. Encara així, en el 2015, per cada 1000 passatgers s'extravien unes 6,53 maletes segons *The Baggage Report 2016*, elaborat per la Societat Internacional de Telecomunicacions Aeronàutiques (SITA) [2]. En valors absoluts, tenim 23,1 milions de maletes perdudes o extraviades en la manipulació i el transport d'aquestes.

En termes econòmics, la incorrecta gestió dels equipatges va suposar una despesa de 2.300 milions d´euros en els comptes de la indústria aeronàutica. Afegint les pèrdues en objectes de valor personal, la insatisfacció per part dels clients, i les posteriors molèsties per poder recuperar els objectes perduts.

En l'actualitat, el sistema predominant per el rastreig de maletes és l'anomenat *WorldTracer*, desenvolupat per la mateixa SITA. Aquest sistema està integrat en més de 2800 membres en tot el món, incloent-hi infraestructures aeroportuàries i operadors del sector.

Pel que respecta a la plataforma, utilitza uns identificadors inscrits en les pròpies *Baggage Tag,* o comunament, anomenades etiquetes de facturació de color verd o fons blanc. Inscrit en una de les cares, figura un codi de barres o símbol QR, que relacionen l'equipatge amb l'identificador llegit.

D´altra banda, tal com s'ha exposat anteriorment, la gran proliferació del sistema ha permès una millora anual en la disminució d'objectes extraviats. No obstant, si analitzem en més detall els informes, tots apunten a una tardança en la devolució mitjana d'uns 2 dies aproximadament. En altres paraules: en el cas d'una pèrdua, subtracció o mal etiquetatge, el temps d'espera abans de la localització i retorn de l'equipatge és de com a mínim 48 hores.

1.1.1. Cas d'estudi

Després de les cancel·lacions i retards en l'operació sortida de l'estiu de 2016, per part de la companyia *Vueling Airlines*, al voltant de dues mil maletes van ser extraviades durant un període inferior a una setmana [3]. Com a conseqüència, l'Aeroport del Prat va haver d'habilitar dos magatzems per a

l'allotjament de l'equipatge extraviat. Finalment es va retornar als seus propietaris en poques setmanes, o bé com en alguns casos, no es va poder retornar mai. Amb aquest cas, es desprèn la necessitat d'oferir un sistema alternatiu capaç d'evitar l'acumulació de maletes. Oferint una solució flexible i eficient.

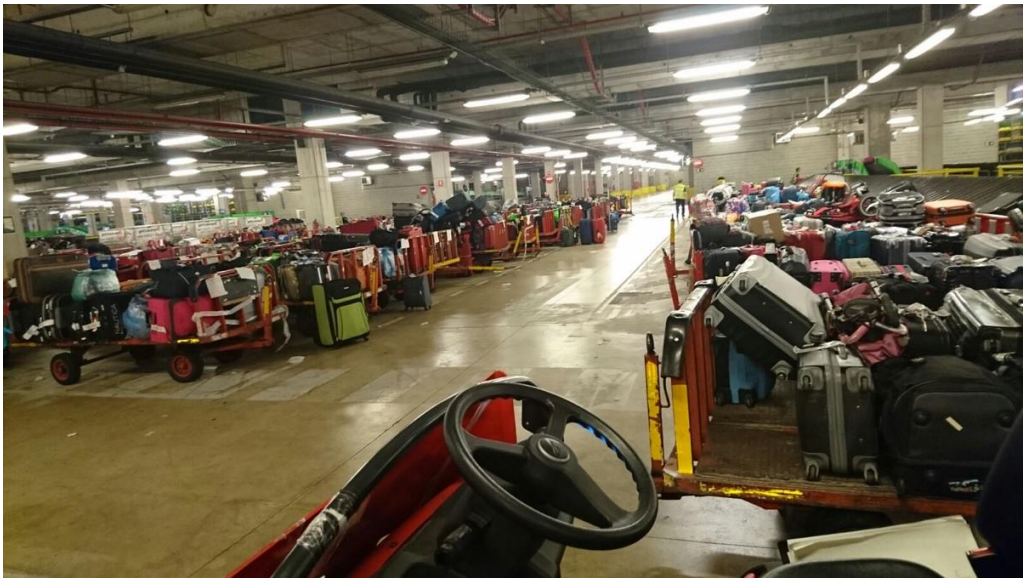


Fig. 1.1. Acumulació de maletes en l'aeroport del Prat. (Juliol 2016)

1.2. Debilitats

En primer lloc, s'ha d'analitzar el perquè actualment hi ha aquestes demores de temps.

La plataforma *WordTracer* està formada per un conjunt de sistemes interconnectats que registren, de forma manual, el trànsit produït per un equipatge en totes les infraestructures i vehicles per on hi ha transitat. En la majoria de casos, es redueix en el punt d'entrada. En exemple, la facturació d'una maleta en el sistema corporatiu de l'aerolínia. Així doncs, l'últim contacte, és el del mateix usuari que factura la maleta en l'aparador de la companyia.

A més a més de la poca gestió, s'ha d'incloure l'ús de lectors portàtils utilitzats per personal de *handling*[1], encarregats de la part logística i el transport d'equipatges fins al seu destí. Aquests lectors compten amb un lector de codis de barres. Així doncs, en cas de pèrdua, l'operari ha de rastrejar els objectes, un per un, per ubicar, en aquest cas, la maleta.

En conclusió, tenim una plataforma actual amb una estructura poc escalable, molt dependent de recursos humans, i poc automatitzada.

[1] *Handling*: és el servei d'assistència a avions en terra és a dir, és el conjunt d'operacions terrestres que permeten que un avió pugui ser carregat i descarregat de passatgers, mercaderia i equipatge.

1.3. Objectiu

L'objectiu del projecte, per tant, és proposar un sistema alternatiu a l'actual. D'una banda el sistema ha de ser suficientment flexible per possibles ampliacions i ben modelat per la gran quantitat d'informació, que ha de ser bolcada dintre de la plataforma. D'altra banda, ha de ser un sistema implementat amb les últimes tecnologies, eliminant la restricció d'escanejar en un camp de visió perfecte els codis de barres. I per últim, que el client tingui la possibilitat de conèixer l'estat del seu equipatge.

Si bé és cert que s'ha hagut d'acotar la grandària del projecte per qüestions merament de profunditat, disseny, aspectes tècnics i àmbit d'extensió del treball de final de grau. No té afectació en el plantejament de la idea, ni en la disminució dels objectius principals del projecte.

Objectius:

- La utilització de clauers RFID per la lectura d'un identificador, que permeti la identificació de la maleta en la base de dades del sistema.
- La creació d'una plataforma web, que permeti la gestió i la introducció de dades per part dels usuaris administradors, i en especial, de les aerolínies.
- La creació d'una aplicació mòbil, que permeti el seguiment de l'estat de l'equipatge i la lectura dels clauers RFID.
- La implementació de mecanismes de seguretat per l'autentificació d'usuaris i la protecció de dades vulnerables.
- La utilització d'una *Raspberry Pi* amb una targeta lectora de clauers RFID. Amb funcionalitat de llegir-ne el contingut emmagatzemat en el clauer.
- L'obtenció de dades de forma semiautomàtica referent a les aerolínies, vols, avions i aeroports, mitjançant diferents fonts d'informació.
- Per últim, la implementació dels objectius anteriorment esmentats, amb les últimes tecnologies de l'estat de l'art actual.

CAPÍTOL 2. ESPECIFICACIONS DEL PROJECTE

En el present capítol s'exposa les especificacions tècniques del projecte. Està dividit en 3 subapartats. Cada punt s'ocupa de definir les funcionalitats, els rols de cada usuari en el sistema, i el hardware necessari per a l'execució dels punts anteriors.

Durant l'especificació del projecte, apareixen dos tipus d'usuaris a escala lògica o descriptius, per ajudar a definir per qui va adreçada la funcionalitat o millorar l'exposició dels rols del sistema:

- En primer lloc, l'usuari gestor, operador o administratiu. Aquest usuari es refereix al personal privat que gestiona el sistema, administrador, o forma part d'un operador, en aquest cas, les aerolínies.
- En segon lloc, un usuari anomenat client que es refereix als usuaris finals. Com usuari final s'entén que és el passatger que n'utilitza la plataforma per al seguiment de l'equipatge com a servei, i no té cap permís per poder realitzar modificacions al sistema.

2.1. Funcionalitats

A continuació, s'exposa una enumeració de totes les funcionalitats necessàries per a la gestió completa i el monitoratge en el seguiment dels equipatges.

2.1.1. Plataforma Web

Pel que respecta a la gestió per la plataforma web:

- Accedir al sistema mitjançant un procés d'autentificació.
- Gestió de comptes d'usuaris. En especial, la creació d'usuaris amb privilegis de gestors o administradors.
- Importació d'informació o dades. Provinents d'APIs, per tècniques de *Web Scraping*[2] o fitxers prèviament carregats al sistema:

    1. Dades respecte a totes les companyies aèries actives.
    2. Flotes d'avions en relació a cada companyia aèria registrada.
    3. Informació de tots els aeroports comercials del món.
    4. Vols/rutes comercials actius/ves per cada companyia aèria registrada.

[2] *Web Scraping: és una tècnica de programari o software informàtic per extreure informació dels llocs web.*

- Gestió de les dades provinents de la pròpia base de dades. Especialment, en la creació, modificació, actualització i eliminació de documents registrats pel que fa a:

    1. Comptes d'usuaris finals
    2. Companyies aèries
    3. Flotes d'avions
    4. Vols i/o rutes comercials
    5. Equipatge registrat

2.1.2. Aplicació mòbil

Quant a l'aplicació mòbil tenim les següents funcionalitats:

- Accedir l'aplicació mitjançant un procés d'autentificació.
- Lectura del clauer RFID ubicat en el cos de la maleta.
- Consulta de l'estat de l'equipatge registrat al compte de l'usuari.

2.2. Rols

El present apartat tracta d'especificar la quantitat de rols disponibles en el sistema. Cada rol compta amb uns privilegis d'accés que permeten la utilització de mòduls o l'ús de les funcionalitats específiques. Per consegüent, cada tipus d'usuari tindrà un ventall d'operacions limitat al rol assignat.

2.2.1. Administrador

El rol d'Administrador és un dels més complets en tot el projecte. És l'únic que pot importar i carregar nova informació a la base de dades. Com el seu nom indica, s'encarrega de realitzar les operacions de manteniment i mantindré les dades en constant actualització. A més a més, és dels pocs usuaris que poden crear comptes amb rols d'administrador o d'operador. Té totes les funcionalitats activades menys l'opció de facturar l'equipatge.

2.2.2. Operador

El rol d'Operador és el definit per designar els usuaris que són personal administratiu de les companyies aèries. Contràriament al rol d'Administrador, pot modificar únicament les dades registrades que fan referència a l'aerolínia. Per tant, són els responsables de gestionar la part de facturació i el seguiment

de l'equipatge. Així doncs, no tenen la capacitat de modificar o crear usuaris amb rols d'operador o d'administrador.

2.2.3. Client

El rol de Client és el últim rol en el nombre de funcionalitats disponibles pel sistema. El propòsit de la seva creació, és la diferenciació respecte als dos rols que s'han exposat en els anteriors subapartats. Aquests comptes són necessaris pel correcte registre de l'equipatge, per tal de tenir una relació directa entre maletes registrades i l'individu propietari d'aquestes.

2.3. Hardware

En el present apartat es vol exposar els requisits tècnics quant a components físics del projecte. El conjunt de hardware està compost per una *Raspberry Pi 3,* un lector *RFID RC522,* un clauer RFID, i un *Smartphone/* telèfon intel·ligent compatible amb la lectura de clauers RFID.

Tanmateix, l'ús del telèfon intel·ligent és degut a la capacitat dels últims models per llegir etiquetes NFC i RFID. Mentre que el RFID és una tecnologia que utilitza ones de ràdio per transferir dades des d'una etiqueta o targeta cap a un lector, sense la necessitat de cap contacte entre ells. El NFC és una tecnologia estandarditzada que té com a propòsit facilitar la interconnexió de dispositius i l'intercanvi de dades [4].

En ambdós casos, tots ells utilitzen el mateix estàndard internacional ISO 14443. És a dir, el clauer que s'utilitza en el projecte treballa amb l'estàndard ISO 14443 però no en fa ús de la tecnologia NFC com a tal.

A continuació, en els següents subapartats es donarà una breu definició de cada element, una explicació del seu funcionament i la seva implicació en el projecte.

2.3.1. Raspberry Pi

Raspberry Pi [5][6] és un ordinador SBC (acrònim en anglès de Single-Board Computer) de baix cost, es podria dir que és un ordinador de grandària reduïda, de l'ordre d'una targeta de crèdit, desenvolupat en el Regne Unit per la Fundació *Raspberry Pi* (Universitat de Cambridge) en 2011, amb l'objectiu d'estimular l'ensenyament de la informàtica a les escoles, encara que no va començar la seva comercialització fins a l'any 2012.

Pel que respecta al projecte, la *Raspberry Pi* té com a finalitat de donar suport al mòdul *RFID-RC52* i connectar-lo amb la plataforma del projecte mitjançant

els ports GPIO[3]. També incorporarà un petit servidor web que exposarà les lectures del mòdul.

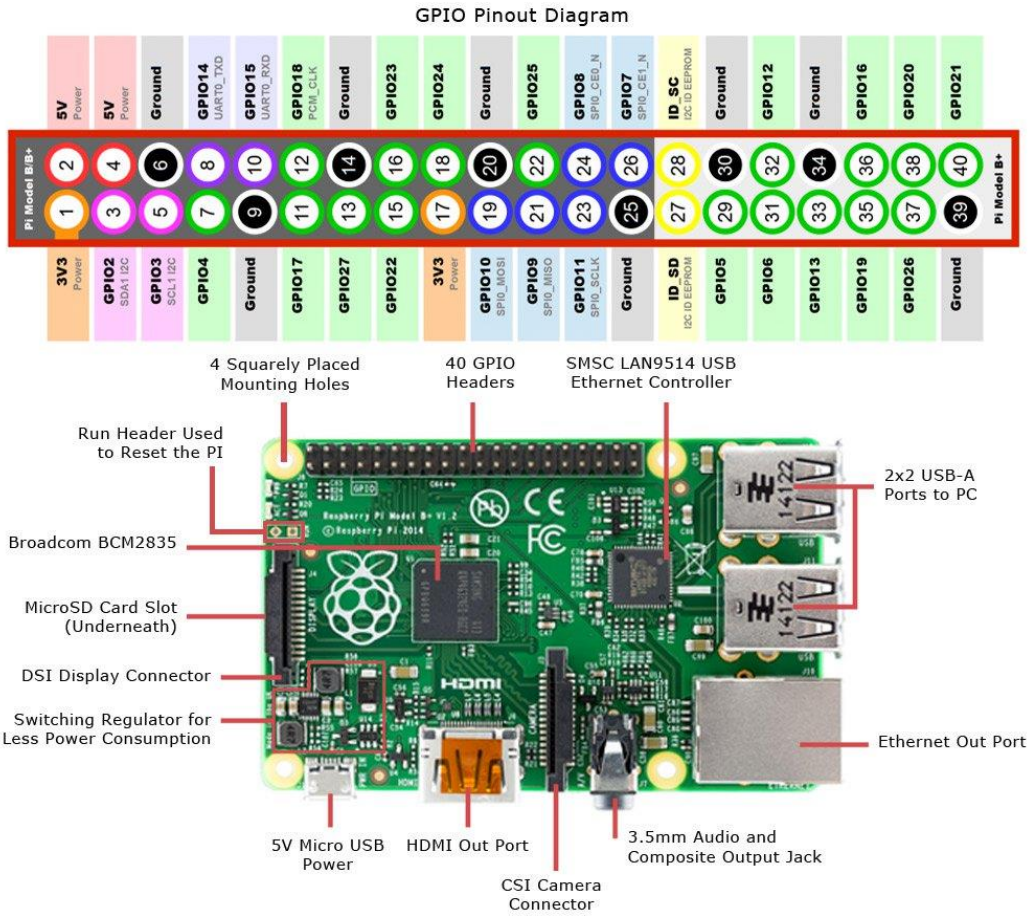


Fig. 2.1. Esquema de connexions d'una Raspberry Pi

2.3.2. RFID-RC522

El RFID-RC522 [7] és un lector format per un microxip integrat per a la comunicació, sense contacte, a una freqüència base de 13.56 MHz. El lector RC522 és compatible amb els estàndards *ISO/IEC 14443 A*, *MIFARE Classic* 1K i *NTAG*.

En altres paraules, el lector està format per una antena inclosa en el propi *PCB* de la placa. A continuació, apareix un conjunt de components electrònics, on s'inclou el microxip de *MFRC522* amb un oscil·lador de quars a 27.120 MHz, coincidint com a divisor de 13.56 MHz. I per últim, una fila de pins per instal·lar una connexió fins als GPIO de la *Raspberry Pi.*

[3] GPIO: és un pin genèric en un xip, el comportament del qual (incloent si és un pin d'entrada o sortida) es pot controlar (programar) per l'usuari.

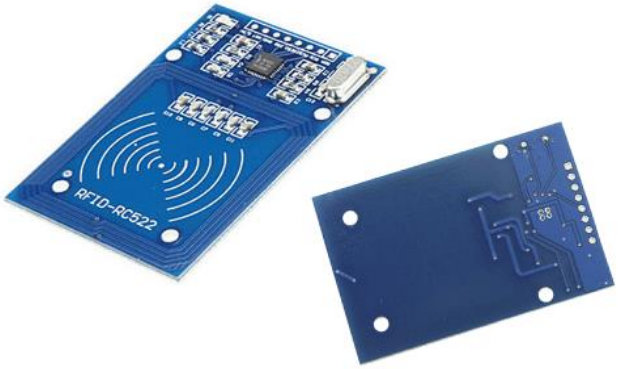


Fig. 2.2. Mòdul MFRC522

Així doncs, compta amb una antena rectangular, dissenyada per obtenir l'energia i habilitar la lectura o l'escriptura de la memòria no volàtil integrada. L'objectiu de l'antena és induir un petit corrent elèctric, suficientment potent, per poder fer funcionar el clauer RFID. Amb l'objectiu de tornar les dades registrades cap a la placa lectora. Per tant, el clauer no té la necessitat d'incloure una bateria, ja que és un element passiu.

En aquesta versió del clauer s'utilitza l'estàndard propietari *MIFARE Classic* 1K de NXP [8].

Fig. 2.3. Clauer MIFARE Classic 1K

La finalitat d'utilitzar el dispositiu en aquest projecte és l'emmagatzematge d'un identificador de l'equipatge. Aquest ID serà llegit per part del lector, que al mateix temps, enviarà la informació mitjançant la connexió amb els ports GPIO.

En el cas del *Smartphone* o telèfon intel·ligent, el propi microxip integrat en el mòbil farà de lector, com qualsevol etiqueta *NFC,* per mitjà d'algun *plugin* o llibreria en l'aplicació.

CAPÍTOL 3. ARQUITECTURA

El present capítol es descriu el disseny de l'arquitectura i les connexions amb altres components. També el model de dades utilitzat per crear la base de dades.

3.1. Arquitectura

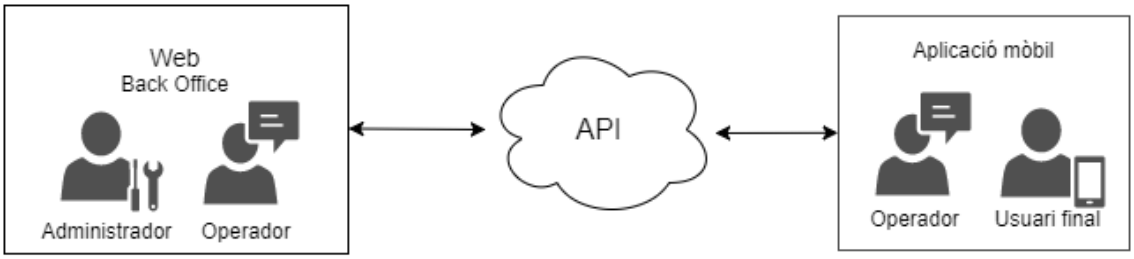


Fig. 3.1. Vista disseny de l'arquitectura

L'arquitectura del projecte és basada en una API central que donà servei tant a l'aplicació, com al portal web o *Back Office*[4]. Seguint l'esquema d'objectius i dels rols d'usuari, l'aplicació web estarà disponible únicament pels usuaris administradors i operadors. Tanmateix, l'aplicació mòbil és l'accés del sistema per part de l'usuari o client final. No obstant, l'operador també té accés per la lectura de clauers in situ.

3.2. Models de dades

Aquest apartat s'ocupa d'analitzar el model de dades implementat en la base de dades NoSQL amb un sistema MongoDB. Cada subapartat es correspon a la definició del model i a les anotacions relacionades amb la finalitat de cada camp en l'esquema o la pròpia definició teòrica.

Una de les particularitats del model de dades és la seva complexitat. Aquesta és deguda a poder utilitzar al màxim les capacitats de la tecnologia de GraphQL, i obtenir un model més realista com es podria esperar en l'àmbit aeroportuari.

Per tal de portar a terme les relacions en un sistema no relacional, es fa ús de documents i subdocuments. Un document és una instància d'un model. És a dir, un document és una entrada en la base de dades on té els camps indicats

[4] *Back Office: comprèn el programari que una organització utilitza per administrar operacions que no estan relacionades amb cap esforç de vendes directes (com un venedor amb un client present) i interfícies que els consumidors no veuen.*

pel model que pertany. En canvi, un subdocument és un document incrustat en un document [9].

Per tant, la forma que té MongoDB per formar relacions entre documents són els subdocuments. Cada cop que un document es crea dintre del sistema, es genera un ID (_id). Quan es vol incloure una relació en un document, s'agafa l'ID del document desitjat, i s'incrusta en el document actual. Posteriorment, s'utilitzen mecanismes per obtenir la informació relacionada amb aquell ID. En la figura següent, cada línia suposa una relació directa, on la punta de la fletxa és el model o conjunt de models incrustats en el model origen.

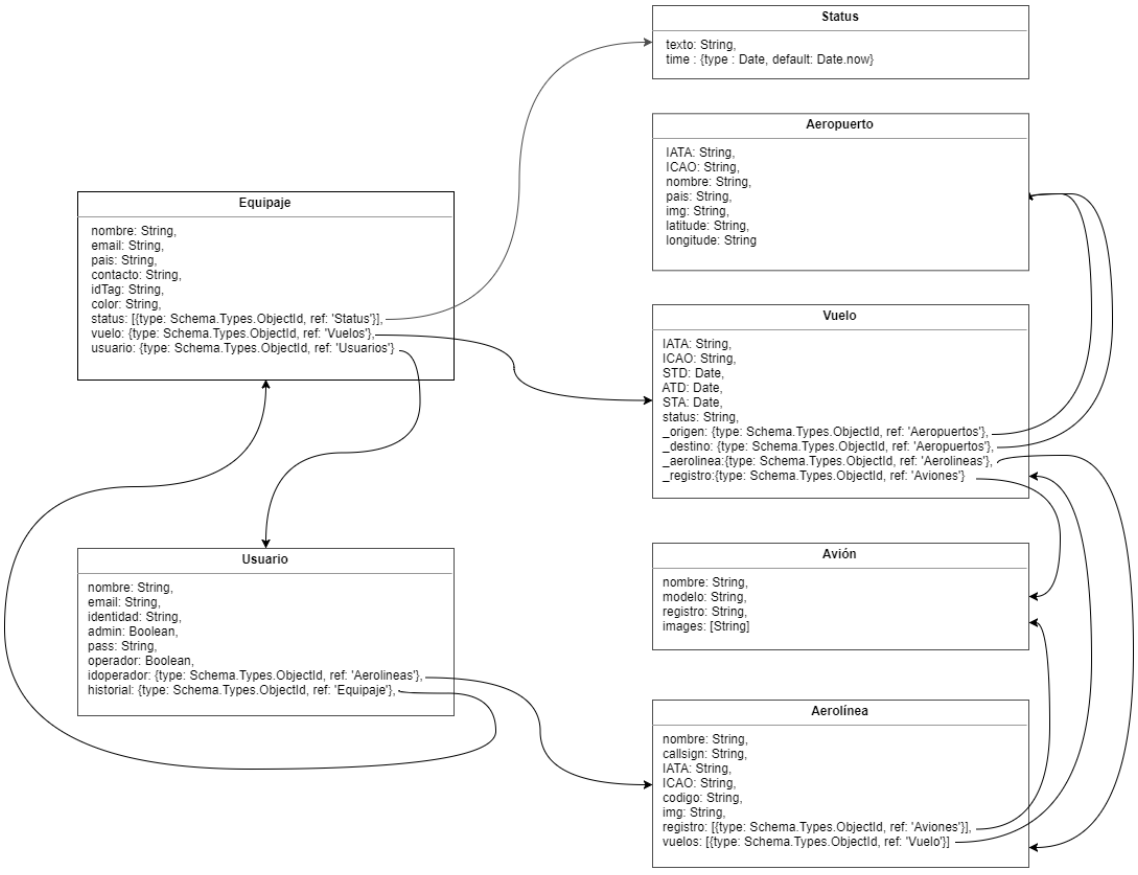


Fig. 3.2. Vista general dels models

3.2.1. Model Equipatge i Model Status

En aquest subapartat es definirà el model Equipatge i el model Status. El model equipatge està dissenyat per registrar dades relatives a l'equipatge. I el model Status per emmagatzemar els estats d'un equipatge. En definitiva:

Model Equipatge:

- Nom, correu electrònic, país, color (de la maleta) i contacte (número de telèfon, altre correu): informació bàsica.
- idTag: identificador que relaciona el propi document de la base de dades amb el clauer RFID.
- Status: conjunt de subdocuments que contenent el model Status. Conté la data i l'estat de l'equipatge. [Subdocument]
- Vol: relaciona l'equipatge amb l'identificador de vol on serà transportat. [Subdocument]
- Usuari: indica l'usuari propietari de l'equipatge. [Subdocument]

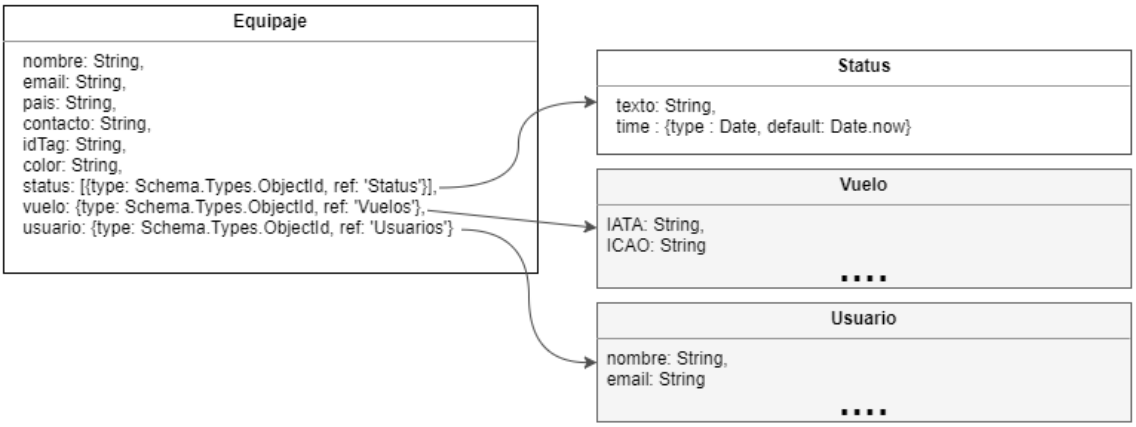


Fig. 3.3. Model Equipatge i Model Status

Model Status:

- Text: indica l'estat de l'equipatge.
- Temps: és una marca de temps (data creació) en relació a l'estat de l'equipatge.

3.2.2. Model Usuari

El model d'Usuari defineix un conjunt de camps necessaris per la identificació de l'usuari en el sistema. En conclusió:

- Nom, email, identitat (NIF, CIF o Passaport): informació bàsica.
- Admin: valor booleà indicatiu si l'usuari és (rol) administrador.
- Operador: valor booleà indicatiu si l'usuari és (rol) operador.
- Pass: clau/contrasenya registrada per l'usuari.
- Idoperador: En cas de ser operador, l'identificador que relaciona de quina aerolínia prové. [Subdocument]
- Historial: conjunt de subdocuments que conté l'historial d'equipatges per part d'un usuari. [Subdocument]

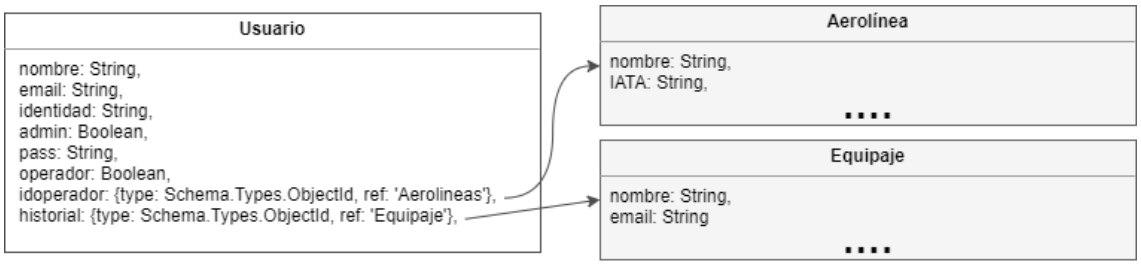


Fig. 3.4. Model Usuari

3.2.3. Model Aerolínia

El model Aerolínia constitueix els camps necessaris amb la finalitat d'obtenir informació relativa a les aerolínies del sistema. En resum:

- Nom: el nom de l'aerolínia.
- IATA: és un identificador IATA[5] per la identificació d'aerolínies en l'àmbit comercial.
- ICAO: és un identificador ICAO[6] utilitzat per la identificació d'aerolínies en gestió del control aeri.
- Callsign; és la combinació del camp IATA i ICAO amb un separador.
- Codi: identificador utilitzat per la importació de dades en la implementació del *Web Scraping.*
- Img: URI del logotip de la companyia.
- Registre: conjunt de subdocuments que conté la totalitat d'avions de propietat per part d'una aerolínia. [Subdocument]
- Vols: conjunt de subdocuments que conté el total de vols registrats de l'aerolínia. [Subdocument]

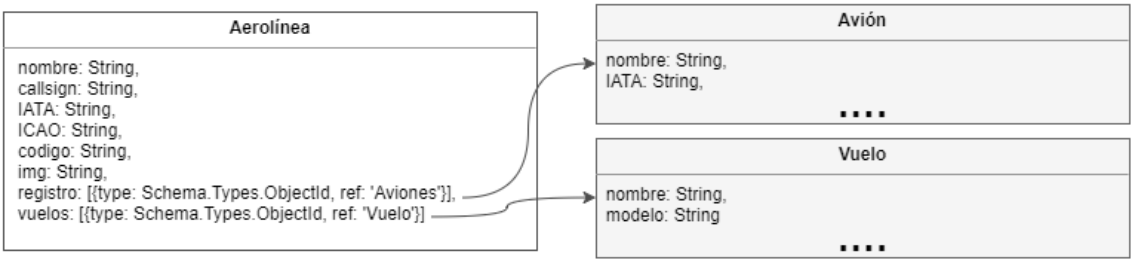


Fig. 3.5. Model Aerolínia

[5] IATA: International Air Transport Association. És una associació comercial que se centra a fer que els negocis de trànsit aeri siguin segurs, segurs, fiables i eficients. Els codis IATA s'utilitzen principalment per a la venda d'entrades.

[6] ICAO: International Civil Aviation Organization. És un organisme de les Nacions Unides que se centra en l'harmonització internacional de les regulacions de l'aviació civil. Els codis de la OACI s'utilitzen per a propòsits "oficials" com el control de trànsit aeri.

3.2.4. Model Avió

El model Avió està dissenyat per l'emmagatzematge d'informació relacionada amb els avions del sistema. En conclusió:

- Nom, model, registre: informació bàsica de l'avió
- Images: llistat d'URIs de fotografies del model d'avió en qüestió.

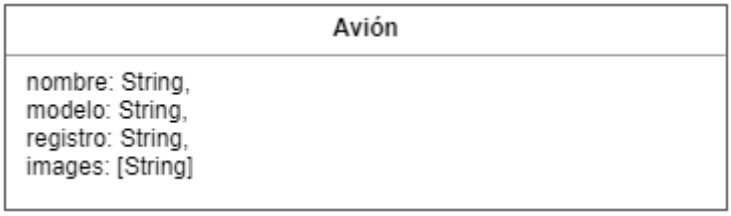


Fig. 3.6. Model Avió

3.2.5. Model Vol

El model Vol defineix un conjunt de valors per l'obtenció de dades relatives als vols del sistema. Conté informació relacionada amb identificadors *IATA/ICAO*, horaris i estat del vol:

- IATA: és un identificador IATA per la identificació de vols en l'àmbit comercial.
- ICAO: és un identificador ICAO utilitzat per la identificació de la ruta realitzada per l'avió.
- STD, ATD, STA: Temps Programat de Sortida, Temps Actual de Sortida, Temps Programat d'Arribada.
- _origen, _destí: identificadors per relacionar la informació dels aeroports relativa a l'origen i destí del vol. [Subdocument]
- _aerolínia: identificador relatiu a l'operador del vol, en aquest cas, el de l'aerolínia. [Subdocument]
- _registre: identificador relatiu a l'avió utilitzat per operar el vol. [Subdocument]

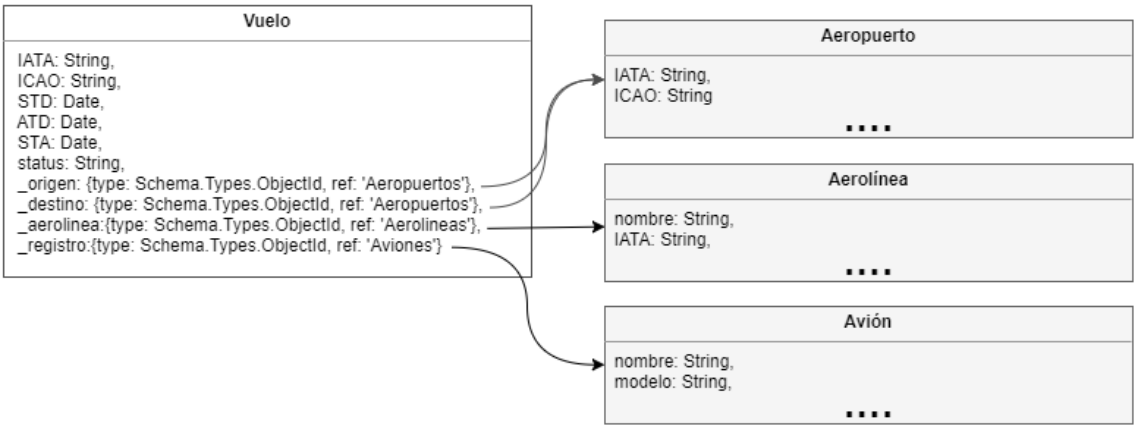


Fig. 3.7. Model Vol

3.2.6. Model Aeroport

El model Aeroport constitueix els camps necessaris per la dotació d'informació relacionada amb els aeroports registrats per el sistema. En conclusió:

- IATA: és un identificador IATA per la identificació dels aeroports en l'àmbit comercial.
- ICAO: és un identificador ICAO utilitzat per la identificació de l'aeroport en la gestió aeroportuària del control aeri.
- Nom, país: informació bàsica relativa l'aeroport.
- Img: URI de la bandera del país on pertany.
- Latitude, longitude: coordenades geogràfiques de la ubicació actual de l'aeroport.

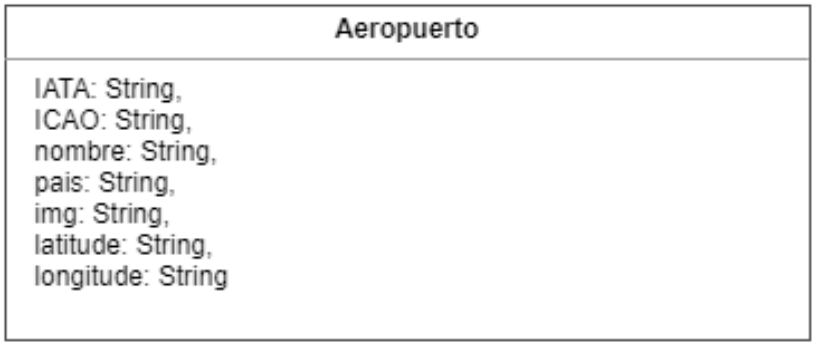


Fig. 3.8. Model Aeroport

CAPÍTOL 4. IMPLEMENTACIÓ

En el present capítol s'exposen definicions, tècniques i la implementació de les últimes tecnologies actuals per la formalització dels objectius principals. El capítol és estructurat en diversos subapartats segons la finalitat de cada component. El nom escollit per denominar el projecte en la implementació és BagTracker**.**

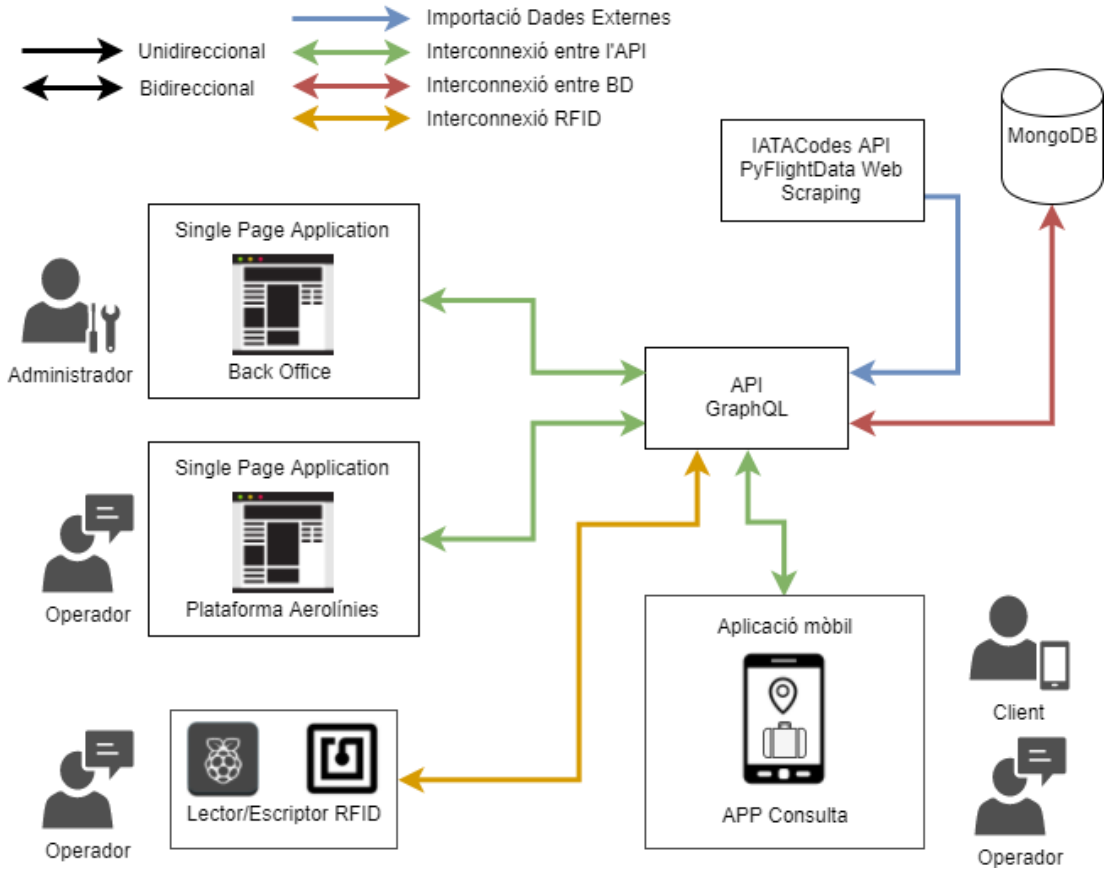


Fig. 4.1. Vista de la implementació

En primer lloc tenim la vista d'implementació on és mostra tots els components que la conformen. El nucli de tota l'arquitectura està basat en la utilització d'una API desenvolupada amb GraphQL. Tanmateix, l'obtenció de les dades referents a l'estat dels vols, informació d'aerolínies o la descàrrega de flotes d'avions es porta a terme per l'ús d'altres APIs externes i/o la utilització d'un mòdul encarregat de realitzar *Web Scrapping*. En el cas de la base de dades s'utilitza una NoSQL amb un sistema MongoDB.

Per la part del client web, hi ha dues configuracions: *Back Office* (Administrador) i la plataforma d'aerolínies. Aquesta és dinàmica i ajusta els continguts segons el rol de l'usuari (apartat 2.2 Rols). Per exemple, el *Back Office* és únicament accessible des d'un compte amb rol d'Administrador, mentre que, pel cas de les aerolínies, s'utilitza el rol d'Operador. En conclusió,

el client és el principal gestor de la base de dades. D'altra banda, hi ha el mòdul lector/escriptor RFID. Té una connexió directa amb l'API *GraphQL* i permet l'obtenció d'informació de forma interna, encara que és un mòdul extern.

Per últim, l'aplicació mòbil permet la consulta de dades relatives a l'equipatge consumint directament de l'API. A diferència del client, el dispositiu mòbil ha d'incloure la capacitat de lectura de clauers RFID.

4.1. Preàmbul

Un dels aspectes problemàtics, a l'hora d'implementar els estats de les maletes, era la grandària del projecte. Tal com s'exposa en els apartats anteriors (apartat 1.3. Objectiu), un dels problemes que hi havia en el desenvolupament previ del projecte, era la creació d'una xarxa de sensors. La finalitat de la xarxa és l'actualització automàtica de l'estat de l'equipatge. En un exemple, es planifica de la següent manera:

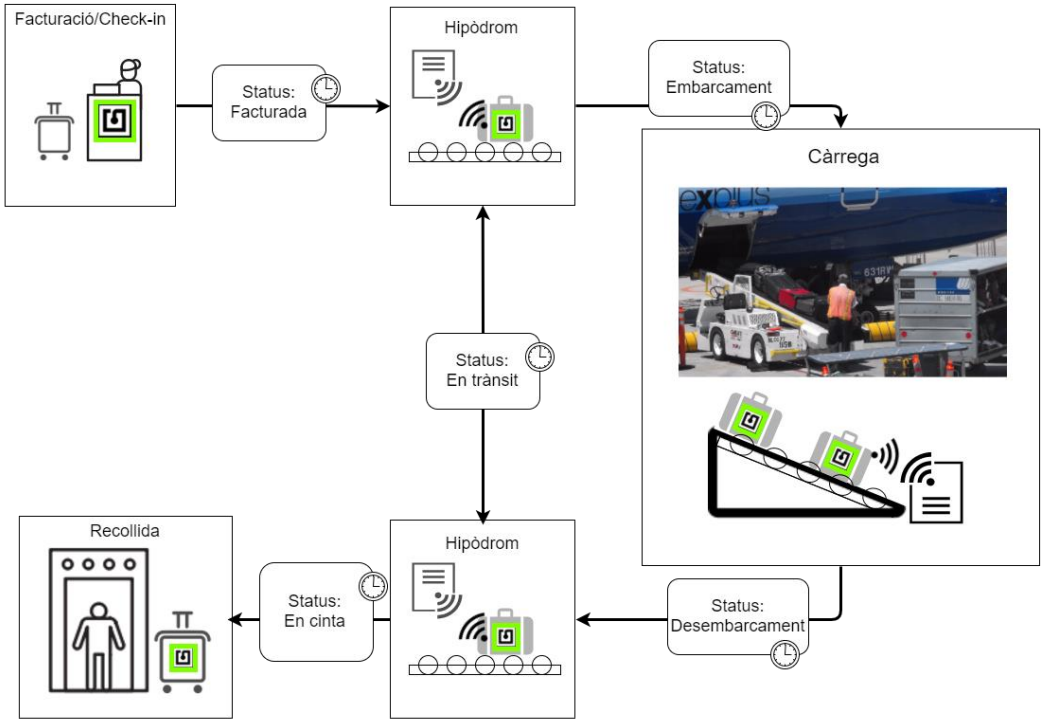


Fig. 4.2. Diagrama d'estats de l'equipatge

Com es pot observar en la figura anterior, cada punt de trànsit de la maleta té el requeriment de comptar amb un lector de clauers RFID. Per tal d'evitar l'ús de múltiples sensors i la seva corresponent configuració, la solució optada per adaptar l'escenari en el projecte és la creació d'un mòdul web anomenat simulador. Aquest mòdul formarà part de la suite de gestió en la plataforma de les aerolínies. En conclusió, s'utilitza el mòdul simulador per evitar la creació d'una xarxa de sensors a causa de no sobrepassar l'àmbit del projecte.

4.2. Servidor

El servidor és l'encarregat de subministrar la informació demanada i ajudar a realitzar canvis en la base de dades. En el cas del projecte, també permet la comunicació directa amb el lector RFID.

Una API (acrònim de l'anglès *Application Programming Interface)* o Web API és un conjunt de recursos exposats que permet realitzar algunes accions, accedir a característiques, o al contingut del servidor mitjançant rutes [10]. Quan parlem de rutes es fa referencià a *endpoints.* Un *endpoint* és una ruta d'accés via URI on al realitzar una petició HTTP es pot rebre una resposta.

En el món del desenvolupament web és habitual la implementació d'una API REST. Una API REST és un estil d'arquitectura que segueix un conjunt de principis com [11]:

- Client/servidor sense estat: al demanar una petició al servidor respon amb una resposta idèntica en totes les peticions idèntiques. Per tant, no necessita recordar cap estat anterior.
- Les operacions més importants són les CRUD: Create (Crear), Read (Llegir), Update (Actualitzar) i Delete (Eliminar).
- Els models de dades únicament són manipulats mitjançant un mètode CRUD i des d'una URI/*endpoint*.
- HATEOAS: és un principi que defineix la necessitat de retornar una resposta amb informació associada a altres recursos per mitjà d'enllaços URI.

Tanmateix, en el projecte no s'utilitzarà l'estil REST per la implementació del servei. Més aviat, s'utilitzarà la tecnologia de GraphQL. En resum, l'API del projecte està composta per quatre components que s'explicaran a continuació:

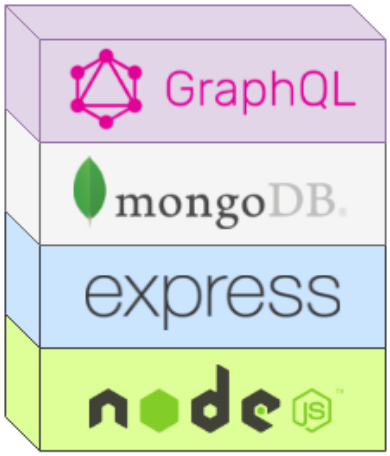


Fig. 4.3. Pila de components

4.2.1. Javascript i ES6

Pel que respecta al llenguatge de programació s'utilitza Javascript en el servidor. En especial la versió del nou estàndard ECMASCript 6.

ECMAScript 6, ES6 o ES2015 [12] és una nova especificació de l'estàndard de Javascript que incorpora una nova sintaxis al definir funcions, classes i mòduls. També inclou noves característiques com els iteradors for/of, creació de mòduls, etcètera. N'és un exemple comparatiu:

```
// ES5
var funcio = function(num) {
        return num + num;
}

// ES6
var funcio = (num) => num + num;
```

Fig. 4.4. Sintaxis d'ES5 respecte ES6

No obstant això, un dels inconvenients actuals és la necessitat d'utilitzar un transpilador. Un transpilador és un compilador que tradueix la sintaxis i les funcionalitats de ES6 cap a la versió ES5. L'objectiu de realitzar el procés és convertir la actual sintaxis en codi compatible amb els navegadors web. Això és degut al fet, que actualment no tots els navegadors executen ES6 o falten components per implementar. Per tant, el seu ús és obligatori.

Un dels transpiladors que s'utilitza en el projecte és el Bable.js [13].

4.2.2. Node.js

Node.js [14] és un entorn d'execució per Javascript construït amb el motor de Javascript V8 de Chrome. Node.js utilitza un model d'operacions asíncrones i orientat a esdeveniments, que ho fa lleuger i eficient. Un punt a favor és l'ecosistema de paquets de Node.js, on es poden instal·lar llibreries de codi obert per afegir noves funcionalitats.

La finalitat d'utilitzar Node.js en el projecte és degut a la fàcil importació dels mòduls necessaris mitjançant el seu propi gestor NPM. A part, és un motor testejat i constantment actualitzat.

4.2.3. Express

Express [15] és un framework per Node.js, minimalista i flexible per Node.js que permet crear un conglomerat de rutes i operacions CRUD de forma simple.
La implementació d'Express en el projecte es redueix en la creació de tres rutes: dos per poder accedir a GraphQL i una per iniciar sessió de l'usuari:

```
app.use(bodyParser.urlencoded({limit: '5mb', extended: true }));
app.use(bodyParser.json({limit: '5mb'}));

const helperMiddleware = [
  bodyParser.json(),
  middleware.isAuth,
];

app.use('/graphql', ...helperMiddleware, graphqlExpress({
      schema: schema
}));

app.use('/graphiql', graphiqlExpress({ endpointURL: '/graphql'}));

app.use('/login', middleware.login);
```

Fig. 4.5. Utilització Express en el servidor Nodejs

Tal com mostra la figura, al contrari d'un servidor amb estil REST, les úniques rutes actives reals són tres. La primera és un *endpoint* d'entrada al servei de GraphQL, mentre que la segona és una ruta per la interfície visual de GraphQL (veure annexos). I l'última és un *endpoint* per iniciar sessió sense passar per GraphQL.

L'objectiu d'aquest últim *endpoint* és evitar incloure un *token* de sessió en la primera petició que s'origini cap al sistema. Si el *token* no és present, el servidor no respon i declina qualsevol petició. Tot plegat amb l'objectiu d'evitar l'ús del servei sense autorització. També, en finalitzar sessió de l'usuari, s'elimina el *token*. Per tant, si es tanca sessió, tornem a l'estat de no tenir un *token* present.

Per últim, l'Express s'ha configurat per admetre peticions amb un *payload* de fins 5mb. Això és degut als errors que han sorgit en el desenvolupament del projecte per l'enviament de dades superiors al límit fixat. (Per defecte < 5mb).

En els aparats posteriors es defineix detalladament que és el servei GraphQL i que és un *token* d'autentificació. La interfície visual de *GraphQL (GraphiQL)* es pot trobar explicada breument en els annexos del document.

4.2.4. MongoDB

MongoDB [16] és una base de dades no relacional (NoSQL) de codi obert. En comptes d'emmagatzemar les dades en taules, MongoDB utilitza un tipus d'estructures de documents amb JSON.

Per tal de comunicar la base de dades amb el servidor, s'utilitza un paquet de modelatge de dades per Node.js. Aquest paquet permet agilitzar la forma d'interactuar amb la base de dades respecte al sistema per comandes de MongoDB. Així doncs, en la implementació d'aquest projecte s'utilitzarà Mongoose.js.

Mongoose.js [17] és un ODM (acrònim anglès d' Object Data Model) que permet la creació de models de dades en MongoDB. Permet l'accés a la base de dades mitjançant funcions *middleware*[7] per agilitzar la comunicació entre el servidor i la base. Un altre de les propietats, és el retorn dels resultats de forma orientada a objectes, per facilitar el mapeig de les dades en l'entorn Javascript.

La implementació de Mongoose.js s'utilitza únicament en el conjunt d'operacions de GraphQL. Una operació definida a GraphQL s'entén com una funció que retorna un valor segons el model de dades prèviament definit. Per exemple:

```
aeropuerto: (obj, { iata }) => {
  return new Promise((resolve, reject) => {
    mongo.Aeropuerto.findOne({IATA: iata}, (err, res) => {
      err ? reject(err) : resolve(res);
    });
  });
},
aeropuertos: () => {
  return new Promise((resolve, reject) => {
    mongo.Aeropuerto.find({}, (err, res) => {
      err ? reject(err) : resolve(res);
    });
  });
},
```

Fig. 4.6. Utilització de Mongoose.js en la implementació de GraphQL

Com s'observa en la figura tenim dues operacions de *GraphQL*: la primera retorna l'aeroport segons el valor passat, mentre que la segona, retorna tot el conjunt d'aeroports de la base de dades. Cada operació està implementada en una *promise* amb la funció de Mongoose.js que genera la solució. Una *promise* o premissa és una funció que espera la finalització del codi contingut dintre d'ella. En el moment que finalitza, la *promise* recull la informació dipositada en

[7] Middleware: és una funció intermèdia que realitza un conjunt d'operacions abans de continuar amb el cicle natural del codi.

la funció *resolve(),* si tot és correcte*.* En cas contrari, retorna l'error amb *reject()*, i informa que la *promise* no ha sigut finalitzada. Això permet convertir l'execució asíncrona de Javascript en una execució síncrona. I generar respostes correctes amb un temps adequat.

En resum, en l'interior de cada premissa hi ha el codi corresponent a les operacions realitzades amb l'ODM Mongoose.js per tal de generar el servei de GraphQL. En apartats posteriors, s'explicarà més detalladament aquest punt en concret.

4.2.5. GraphQL

GraphQL [18][19] és un llenguatge de consulta per APIs que permet la resolució de peticions dirigides a l'obtenció d'informació i modificació de dades. A part, GraphQL proporciona una descripció completa i comprensible de les dades emmagatzemades en el model de l'API. Donà el poder als clients per poder demanar la informació que únicament necessitin. També permet l'evolució de l'API sense realitzar noves versions, evitant malgastar recursos i temps per habilitar la compatibilitat en clients que utilitzin versions anteriors.

Si es fa una breu comparació amb l'estil REST contra la implementació de GraphQL, es troba:

- En REST hi ha la necessitat d'habilitar tantes rutes URIs com la quantitat de recursos implementats. És a dir, no es pot realitzar múltiples operacions sense afectar a l'escalabilitat i l'evolució de l'API. Per contra, GraphQL ofereix totes les operacions des d'un sol *endpoint*. Sense influir en el nombre de recursos implicats.
- En l'obtenció de dades es pot necessitar fer múltiples peticions a l'API REST per obtenir diverses parts d'informació relacionada. En GraphQL es crea una *query* amb el contingut que es vol obtenir. De manera que s'obté únicament la informació requerida i en un únic procés.
- Per contra, una API REST pot ser consumida per qualsevol implementació que gestioni els mètodes CRUD. Mentre que GraphQL es necessita una implementació, en forma de client, que gestioni la connexió amb l'API GraphQL.
- GraphQL disposa d'un fort sistema de validació. En altres paraules, qualsevol consulta o modificació es verifica que el tipus de valor sigui el correcte. Entre altres, es comprova que la petició té l'estructura i la sintaxis adequada.
- Per últim, GraphQL ofereix una interfície gràfica de l'API. Anomenada GraphiQL, ofereix una documentació de l'API automàticament creada, informació sobre l'estructura dels models de dades actuals, i un llistat de totes les operacions disponibles (veure annexos per més informació).

4.2.6. Tipus de peticions de GraphQL

En GraphQL existeix una estructura a l'hora de diferenciar quin tipus de petició es vol realitzar. En resum hi ha dos tipus d'operacions en un servei GraphQL:

- Query: es defineix com una operació que únicament retorna una informació segons els paràmetres definits en la query. Retorna un Observable[8]. Exemple d'una operació query:

Fig. 4.7. Exemple d'una operació query

- Mutation: es defineix a una operació amb la finalitat de modificar algun valor del model de les dades. De manera similar, primer s'inscriuen els camps a modificar, i després, els paràmetres que es volen recuperar en executar l'operació. Retorna un Observable. Exemple d'una operació mutations:

Fig. 4.8. Exemple d'una operació mutation

[8] Observable: és una variable/mecanisme que manté un estat vigilància continuada. L'entorn no ha esperar a completar la resposta. Per tant, sempre s'està alerta sobre els canvis en la resposta [20].

4.2.7. Apollo Server

Es pot implementar un servei de GraphQL amb diverses llibreries segons el llenguatge de programació escollit. En el cas de Javascript s'ofereix dos possibilitats principals:

- GraphQL.js [21] és la implementació de referencia d'un servidor GraphQL. Per contra, la sintaxis per definir les operacions és complexa. En la figura següent, s'exposa l'estructura a l'hora de definir una sola operació amb GraphQL.js:

```
var schema = new GraphQLSchema({
  query: new GraphQLObjectType({
    name: 'RootQueryType',
    fields: {
      hello: {
        type: GraphQLString,
        resolve() {
          return 'world';
        }
      }
    }
  })
});
```

Fig. 4.9. Definició d'una operació *query*

Implementació amb GraphQL.js: Inicialment es crea un schema nou. Un schema és un esquema/llistat que agrupa totes les operacions de GraphQL. El schema té usualment dos seccions: un objecte query i un objecte mutation; mentre que en aquest cas, únicament es defineix una query amb nom 'RootQueryType'. Dintre de la query 'RootQueryType' hi ha l'operació hello. L'operació té com a retorn una variable GraphQLString, que és el format/nom de GraphQL per la definició d'una variable *string*. Això té com a objectiu validar la resposta com correcta, en cas que la base de dades retornés un valor booleà o numèric.

Per últim, l'operació definida com a hello té definida una funció resolve(). Aquesta funció executa el codi dintre de les claus. En resum, quan algun client de GraphQL realitza una petició demanant l'operació hello, es retorna el *string* 'world'.

Com s'observa, hi ha una gran quantitat de codi estructurat per tal de realitzar una simple operació. En conclusió, la implementació de referencia de GraphQL queda descartada per la seva complexitat a l'hora d'implementar.

- Apollo Server [22] és la implementació de la comunitat Apollo per la producció de GraphQL de forma ràpida i dinàmica. Funciona amb qualsevol *schema* realitzar per GraphQL.

  Les millores d'Apollo Server respecte a la llibreria de referència són: el fet de poder enviar peticions de diverses formes, separar les rutes entre GraphiQL (interfície gráfica) i el servei normal, millores a nivell de protocol HTTP, i una millor compatibilitat amb el client proporcionat per Apollo (Apollo Client).

  També, un dels beneficis en la utilització d'aquesta llibreria és la integració directa amb l'Express del servidor. En la figura a continuació, s'exposa l'ús d'Apollo Server en el projecte:

```
import { graphqlExpress, graphiqlExpress } from 'graphql-server-express';

app.use('/graphql', graphqlExpress({
      schema: schema
}));
```

Fig. 4.10. Implementació amb Apollo Server per Express

En conclusió, el projecte utilitza el paquet graphql-server-express per la implementació de l'API GraphQL. Aquest és específic pels servidors que treballen amb Express. Per tant, l'elecció d'Apollo Server és degut a la facilitat d'habilitar tot el servei de l'API, automàticament, en un *endpoint* d'Express indicant el *schema* que es vol utilitzar. En els següents apartats s'exposa com crear el *schema,* ja que en aquest cas, la creació no la fa el paquet graphql-server-express i es genera de forma externa amb un altra llibreria.

4.2.8. GraphQL Schema Language

Per tal de millorar l'aspecte visual i garantir una major fluïdesa al desenvolupar un *schema*, la comunitat de GraphQL va dissenyar un llenguatge d'escriptura per la definició de *schemas* de forma més dinàmica i àgil.

GraphQL Schema Language [23] és una anotació abreviada per expressar de forma bàsica un *schema* i definir el sistema de variables. En apartats posteriors s'exposaran exemples provinents de la implementació realitzada en el projecte. En els annexos es pot trobar una infografia amb totes les anotacions per GraphQL a mode de resum.

4.2.9. Apollo Tools

Anteriorment s'ha definit com és GraphQL, quin tipus de peticions es poden realitzar, com es defineix un *schema* bàsic amb GraphQL.js i com s'implementa un *schema* creat amb Apollo Server. En aquest apartat s'exposa com s'ha generat el *schema* del projecte utilitzant el GraphQL Schema Language i l'Apollo Tools.

Apollo Tools [22] és un paquet per Node.js que permet la creació d'un *schema* utilitzant el GraphQL Schema Language.

```
const schema = makeExecutableSchema({
  typeDefs,
  resolvers
});

app.use('/graphql', graphqlExpress({
      schema: schema
}));
```

Fig. 4.11. Implementació amb Apollo Tools

Així doncs, en la figura anterior, la constant 'schema' està igualada a la funció makeExecutableSchema d'Apollo Tools.

Aquesta funció és l'encarregada convertir els objectes indicats en un *schema* nou que pugui ser utilitzat per Apollo Server. Per tant, la funció agafa dos valors prèviament importats, que corresponen: a una definició del *schema* creat segons el model i funcionalitats del projecte, i al conglomerat d'operacions definides en el *schema*.

Amb l'objectiu de crear l'objecte i guardar-lo en la constant 'schema' per ser utilitzada en l'*endpoint* d'Apollo Server.

A continuació s'exposen el contingut de les dues variables importades en la funció:

```
const typeDefs = `

 type Equipaje {
  _id: ID,
  nombre: String,
  email: String,
  pais: String,
  contacto: String,
  idTag: ID,
  color: String,
  status: [Stat],
  vuelo: Vuelo,
  usuario: Cuenta
}

 type Query {
  equipaje_id(id: ID): Equipaje
  equipajes: [Equipaje]
}

 type Mutation {
  eliminar_equipaje(id: String): Mensaje

}

 schema {
    query: Query
    mutation: Mutation
}
```

Fig. 4.12. Definició de typeDefs

Objecte typeDefs: Com mostra la figura anterior, primerament es defineix un model de dades utilitzant la clau ‘type’ per cada model que existeixi. Dintre de ‘type’, cada valor del model ha de tenir un tipus de variable o bé, en cas de relació, subdocuments de MongoDB amb el nom del model de dades que fa referència. L’ús dels claudàtors ([ ]) s’utilitza per destacar que el valor és un *array* del model en qüestió. N’és un exemple el camp status on retorna un conjunt d’estats.

Per una altra banda, hi ha les definicions de les operacions *Query* i *Mutation.* Ambdós serveixen per implementar el nom de les operacions, les variables requerides i en la finalització, el model que retornen. En especial: equipaje_id (nom operació), (id: ID) (variables demanades) i Equipaje (retorn model equipatge). Per tant, en el ‘type’ query hi figuren les operacions d’obtenció de dades, i en el ‘type mutation, les operacions per modificar els models.

Per finalitzar, la definició ‘schema’ exposa el dos ‘types’ que recullen el conjunt d’operacions per part de query: Query (com type Query) i mutation: Mutation (com type Mutation).

```
const resolvers = {
  Query: {
    equipaje_id: (obj, { id }) => {
      return new Promise((resolve, reject) => {
        mongo.Equipaje.findById(id, (err, res) => {...};
        });
      });
    },
  },
  Mutation: {
    crear_equipaje: (obj, {params}) => {
      return new Promise((resolve, reject) => {...};
          nEquipaje.save((err, res) => {
            err ? reject(err) : resolve(res);
          });
        });
      });
    },
  },
  Equipaje: {
    status: (Equipaje) => {
      return new Promise((resolve, reject) => {...}.Equipaje.findOne({ _id: Equipaje._id }
        ).populate('status').exec(function(err, res) {
            err ? reject(err) : resolve(res.status);
          });
      });
    },
    vuelo: (Equipaje) => {...},
    usuario: (Equipaje) => {...},
  },
};
```

Fig. 4.13. Definició de resolvers

Objecte resolvers: Pel que fa a la resolució de les operacions, s'exporta la constant 'resolvers'. L'objectiu d'aquesta constant és la definició de què cal fer en cada operació inscrita en l'objecte typeDefs anterior. De forma col·loquial, el typeDefs és el llistat de receptes de cuina, i el resolvers les indicacions per la preparació d'aquestes. Així doncs, s'aconsegueix separar en dues parts la definició de les operacions i el codi d'execució. Si es compara amb la implementació de referència, hi ha una millor organització en la forma d'estructurar el codi i una millor escalabilitat.

En una primera vista de l'objecte es defineix les operacions Query, i després, les modificacions Mutation. Com es va comentar en l'apartat de MongoDB, cada operació inclou un codi, dintre d'una *promise,* per retornar la solució segons la petició realitzada. Per tant, la part principal de l'objecte resolvers és dedicada a emmagatzemar els codis d'execució per totes les operacions a un primer nivell.

El segon nivell d'aquest resolver és la resolució dels models de dades que inclouen subdocuments/documents incrustats. Si s'agafa com a referència el *type* Equipatge, en el camp 'status', necessita obtenir el conjunt de dades relacionades amb el model status. Amb la funció *populate()* de mongoose es recupera els documents per el seu identificador de MongoDB. I per tant, es pot incrustar cada solució obtinguda del valor status en el conjunt de dades que es retornen com a resposta.

La manera d'implementar el procés anterior és definint una nova entrada en el resolvers. Aquesta entrada té el nom, en el cas de la figura, de *type* Equipatge, i dintre de les claus i figuren tots aquells camps que necessiten una operació extra. Com a exemple, el llistat de passos a continuació:

```
query {
   equipaje_id(id: '1234'){
      nombre,
      status{
         texto,
         time
      }
   }
}
```

Fig. 4.14. Exemple d'ús d'una query

Execució de la *query* com a exemple:

1. Inicialment GraphQL validarà si la *query* no té defectes en la seva sintaxi o estructura.
2. En ser una operació *query* es buscarà, pel nom, l'operació a executar.
3. S'executa l'operació. Per tant, s'utilitzarà la implementació de Mongoose.js per obtenir els resultats. Però en aquest cas, el *schema* especifica que el camp 'status' prové d'un altre model de dades.
4. Com s'està fent una petició que retorna el tipus Equipatge, GraphQL anirà a buscar en la definició de 'Equipaje', l'operació amb el nom de 'status'.
5. Un cop dintre, de la mateixa forma que la primera operació, s'obtindrà el conjunt d'estats amb la funció *populate*() de mongoose.
6. Per últim, s'incrustarà la informació del camp 'status' en la resposta i es retorna tot el resultat conjuntament.

4.3. Importadors

Els importadors són els encarregats d'obtenir informació segons el tipus de dades on s'estigui treballant. En el cas del projecte, tots són ubicats en la part de *resolvers* de GraphQL. A nivell d'implementació, hi ha un importador que fa peticions a una API externa com IATACodes.org mitjançant HTTP. L'altre, utilitza una tècnica anomenada *Web Scraping.*

4.3.1. Importador IATACodes.org

L'importador d'IATACodes s'utilitza per obtenir la informació relativa a les rutes comercials d'arreu del món. També en cas de manca de dades per un model, s'utilitza per autocompletar els valors no existents.

A continuació tenim la figura d'exemple on es realitza la petició amb el mòdul *fetch* a l'API d'IATACodes.org. Per aquest cas, és una descarrega de totes les rutes actives comercials d'avions. Per la utilització de l'importador és obligatori l'ús d'una api_key.

```
iatacodes_rutas: (objs, {}) => {
  return new Promise((resolve, reject) => {
    const url = 'https://iatacodes.org/api/v6/routes?api_key='+ iatacodekey;
    fetch(url, { agent:new https.Agent({rejectUnauthorized:false})})
      .then((resp) => resp.json()) // Transform the data into json
      .then(function(data) {
        while(true){
          if(data){
            resolve({json: data.response});
            break;
          }
        }
      });
  });
},
```

Fig. 4.15. Exemple d'ús d'IATACodes.org

4.3.2. Importador Web Scraping

Un dels problemes a l'hora de desenvolupar el projecte és la manca d'API obertes que ofereixin dades actualitzades. Un mecanisme per obtindre informació és el *Web Scraping*.

Web Scraping [24] és una tècnica de programari o software informàtic per extreure informació dels llocs web. En altres paraules, s'utilitza una llibreria que simula l'exploració web d'una persona. Aquesta realitza una anàlisis de totes les dades mostrades i les indexa.

Així doncs, el projecte fa ús d'una implementació anomenada pyflightdata [25]. Pyflightdata és un projecte obert allotjat en un repositori Github amb l'objectiu de recuperar dades de la web flightradar24.com. En el cas del projecte, s'ha realitzat un *fork* degut a l'estat desactualitzat i la necessitat d'incorporar noves funcionalitats.

Pyflightdata utilitza el llenguatge de programació Python i està construït a partir de la biblioteca BeatifulSoup. BeatifulSoup [26] és un conjunt de llibreries i mètodes utilitzats per realitzar *Web Scraping.* No obstant això, en el projecte s'implementen un conjunt d'arxius python que executen funcions prèviament definides. En altres paraules, no hi ha contacte directe amb BeatifulSoup. Un exemple de la utilització d'un script python:

```
from pyflightdata import *
import json
import sys

data = get_fleet(str(sys.argv[1]))
json = json.dumps(data)
print json
```

Fig. 4.16. Exemple d'un fitxer Python utilitzant pyflightdata

Tal com es pot observar, en l'inici del fitxer s'importa la llibreria pyflightdata que conté les funcions com 'get_fleet'. Aquesta funció realitza un *web scraping* per obtenir dades relacionades amb un argument, en aquest cas, el codi d'una aerolínia. Així doncs, retornarà tota la informació de la flota d'una aerolínia en format JSON i la presentarà per pantalla.

Per tal d'obtenir dades i poder executar fitxers amb codi Python en Node.js s'importa la llibreria *python-shell.* Python-shell [27] és una llibreria que permet l'ús de *scripts* Python i establir una comunicació amb Node.js. En especial es destaca l'habilitat de passar paràmetres i recuperar missatges. En el projecte s'ha implementat de la següent forma:

```
fr24_vuelos_aerolinea_codigo: (obj, {codigo_aerolinea}) => {

  return new Promise((resolve, reject) => {
    PythonShell.run('PythonRutasactivasCodigoAerolinea.py',
      {
        mode: 'text',
        pythonOptions: ['-u'],
        scriptPath: '../../Python',
        args: [codigo_aerolinea]
      }, (err, results) => {
        if (err) throw err;
        var parse = JSON.parse(results);
        err ? reject(err) : resolve({json: parse});
      });
  });
},
```

Fig. 4.17. Implementació de pyflightdata

En un primer lloc, es declara una variable amb el conjunt d'opcions disponibles per la llibreria. Les opcions més destacades són:

- Mode: configura el tipus de dades retornades després de l'execució del script. En el projecte: mode: 'json'
- scriptPath: indica el directori on es localitza el *script*.
- Args: mitjà per introduir els arguments que s'utilitzaran en el *script.*

En segon lloc s'executa el *script* indicat amb la variable opcions i els arguments passats. El resultat es retorna en el *callback* de *python-shell*, i per últim, es processen les dades per enviar-les, més tard, com a resposta. Les operacions de l'API GraphQL amb el prefix 'fr' utilitzen aquesta tecnologia per obtenir la informació pel projecte.

4.4. Client

El client és l'eina per la gestió integral del projecte i proposa funcionalitats als usuaris autentificats per la modificació de les dades emmagatzemades en la base de dades. Tal com s'ha descrit en capítols anteriors, cada rol compta amb uns privilegis d'accés que permeten la utilització de mòduls o l'ús de les funcionalitats específiques. Per consegüent, cada tipus d'usuari tindrà un ventall d'operacions limitat al rol assignat.

4.4.1. Disseny

L'esquema de disseny utilitzat en el projecte és el *SPA* (acrònim de l'anglès *Single Page Application).* Una *SPA* [28] és una aplicació que es pot incloure en una sola plana web amb l'objectiu d'aconseguir una interfície d'usuari semblant a les aplicacions clàssiques de PC.

Un dels objectius del client és el desenvolupament web responsiu. En altres paraules, l'ús de tècniques per fer possible l'adaptació visual del projecte a qualsevol dispositiu o pantalla. En la implementació s'optà per utilitzar el paquet *ngx-boostrap.* Una adaptació de la llibreria *Bootstrap* per a clients amb tecnologia Angular. Boostrap [29] és un framework amb la finalitat d'oferir components prèviament dissenyats com tipografies, formularis, menús...

Per últim, en el projecte s'importa una plantilla de *back-office* anomenada CoreUI [30]. Uns dels avantatges d'utilitzar un *template* és partir d'una base ja estructurada i funcional. Durant el desenvolupament del projecte s'ha anat modificant i afegint mòduls segons les circumstàncies demanades per les funcions. En la següent figura es pot observar un diagrama de les diferents vistes implementades per tal d'incloure totes les funcionalitats demanades:

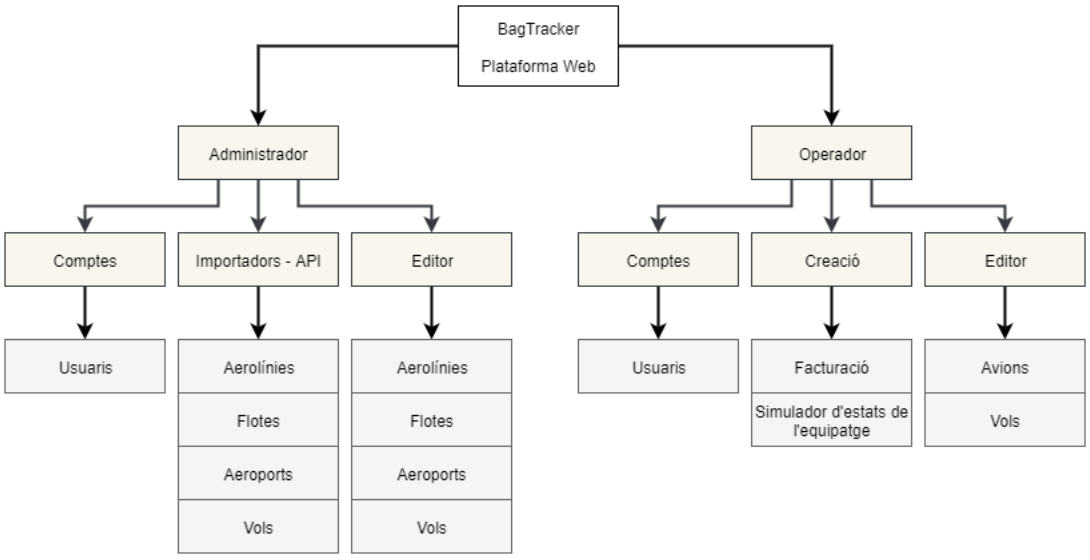


Fig. 4.18. Diagrama de vistes implementades en la plataforma web

Exemple de BagTracker amb el disseny *SPA* amb CoreUI:

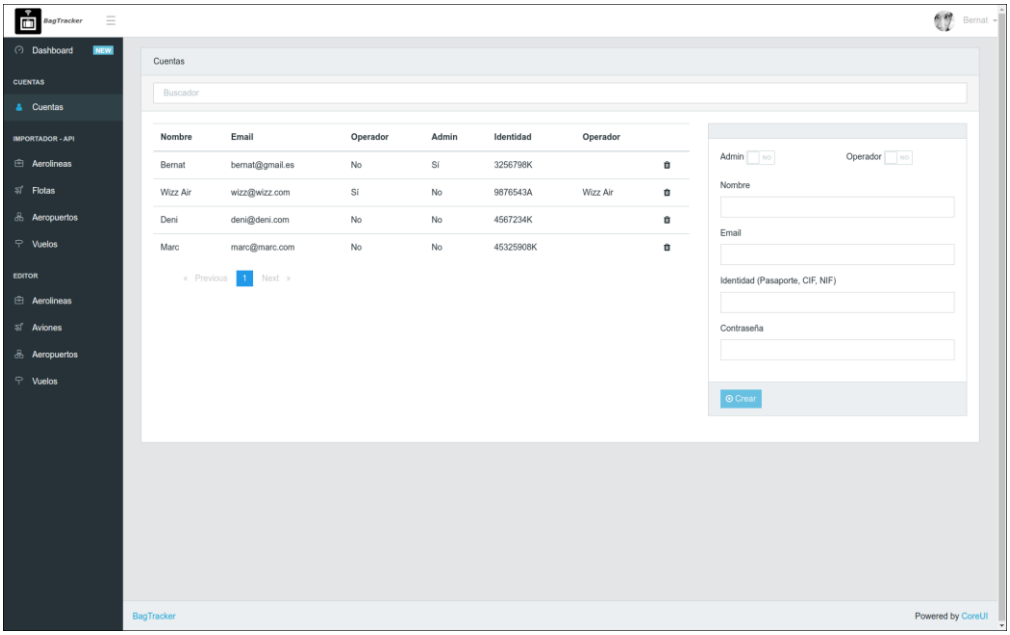


Fig. 4.19. BagTracker amb el disseny *SPA* amb CoreUI

4.4.2. TypeScript

El desenvolupament del client està realitzat amb Typescript. Typescript [31] és un llenguatge de programació lliure i de codi obert desenvolupat i mantingut per Microsoft. Bàsicament Typescript amplia la sintaxis utilitzada en Javascript ES6 o ECMAScript 6 i inclou noves funcionalitats. Els fitxers Typescript tenen l'extensió '.ts'.

De la mateixa manera que Javascript ES6, Typescript necessita un compilador per traduir la sintaxi a una versió de Javascript compatible amb els navegadors webs. En canvi, el compilador va inclòs en la instal·lació del llenguatge, a diferencia d'ES6 que utilitza un compilador de tercers.

4.4.3. Angular 4

Pel que respecta a la tecnologia del client web s'utilitza el *framework* Angular en la seva versió 4.

Angular [32] és un *framework* de codi obert per a aplicacions web desenvolupades en TypeScript, mantingut per Google, on s'utilitza per crear i mantenir aplicacions web d'una sola pàgina o SPA (acrònim de l'anglès *Single Page Application)*.

Angular s'estructura amb contenidors anomenats *Components* que controlen una part de la pantalla mostrada. Cada component té un fitxer amb prefix .ts

que actua com un controlador i un altre .html que funciona com un tros de vista amb llenguatge HTML.

Fig. 4.20. Esquema de components apilats en una SPA

Així doncs, el projecte s'estructura com un collage de components a diferents nivells del SPA. A part dels components, hi ha un arxiu *module.ts* i un de *routing.ts.* Aquests són necessaris per la càrrega de tots els submòduls del projecte, i realitzar un mecanisme de *routing* per gestionar els components que s'utilitzen en el SPA. En l'exemple a continuació, s'exposa com està creat el mòdul editor, encarregat d'implementar totes les funcionalitats per gestionar el sistema des del rol Administrador.

```
@NgModule({
  imports: [
    CommonModule,
    EditorRoutingModule,
    FormsModule,
    NgxPaginationModule,
    Ng2SearchPipeModule,
    ModalModule.forRoot(),
    FillHeightModule,
    TabsModule,
    OrderByModule,
    SlimLoadingBarModule.forRoot(),
    TimepickerModule.forRoot(),
  ],
  declarations: [AerolineasComponent, FlotasComponent, AeropuertosComponent, VuelosComponent]
})
export class EditorModule { }
```

Fig. 4.21. Implementació del mòdul Editor

El fitxer *module.ts* s'encarrega d'importar tots aquells paquets, submòduls i llibreries necessàries per la utilització de funcions en els components i afegir noves funcionalitats. També, en ser un mòdul, ha d'incloure els components que el conformen. En el mòdul Editor de la figura, inicialment es carregen tres submòduls d'Angular. El CommonModule: per habilitar el propi funcionament del mòdul, EditorRoutingModule: el *Routing* associat al mòdul, i FormsModule: necessari per poder crear formularis en els components. Els altres provenen de submòduls creats en el projecte o importats amb el gestor de paquets NPM.

Per últim, consta l'entrada 'declarations' on s'indiquen tots els components associats al mòdul.

```
const routes: Routes = [
  {
    path: '',
    data: {
      title: 'Editor'
    },
    children: [
      {
        path: 'aerolineas',
        component: AerolineasComponent,
        data: {
          title: 'Aerolineas'
        }
      },
      {
        path: 'flotas',
        component: FlotasComponent,
        data: {
          title: 'Flotas'
        }
      },
      {
        path: 'aeropuertos',
        component: AeropuertosComponent,
        data: {
          title: 'Aeropuertos'
        }
      },
      {
        path: 'vuelos',
        component: VuelosComponent,
        data: {
          title: 'Vuelos'
        }
      }
    ]
  }
];
```

Fig. 4.22. Implementació del routing per el mòdul Editor

Per l'altra banda, el fitxer *routing.ts* és l'encarregat de definir les rutes que té el model. És a dir, quan estiguem navegant per el model Editor, la ruta per accedir al component aerolínies és /aerolineas. D'aquesta manera el router d'Angular farà canviar el component actual segons la ruta especificada.

Si es visualitza la figura posterior, des de el punt d'inici o arrel, hi han tres rutes principals. La ruta /admin donà accés als a tots els mòduls per la gestió administrativa del projecte. La ruta /operador suporta els mòduls relacionats a l'administració de les dades de l'operador. I per últim, la ruta /pages té dues funcionalitats. La ruta /login és el primer component carregat en l'inici del projecte per ajudar a l'usuari a autentificar-se. Mentre que la ruta /simulador simularà el pas de l'equipatge per les zones de trànsit.

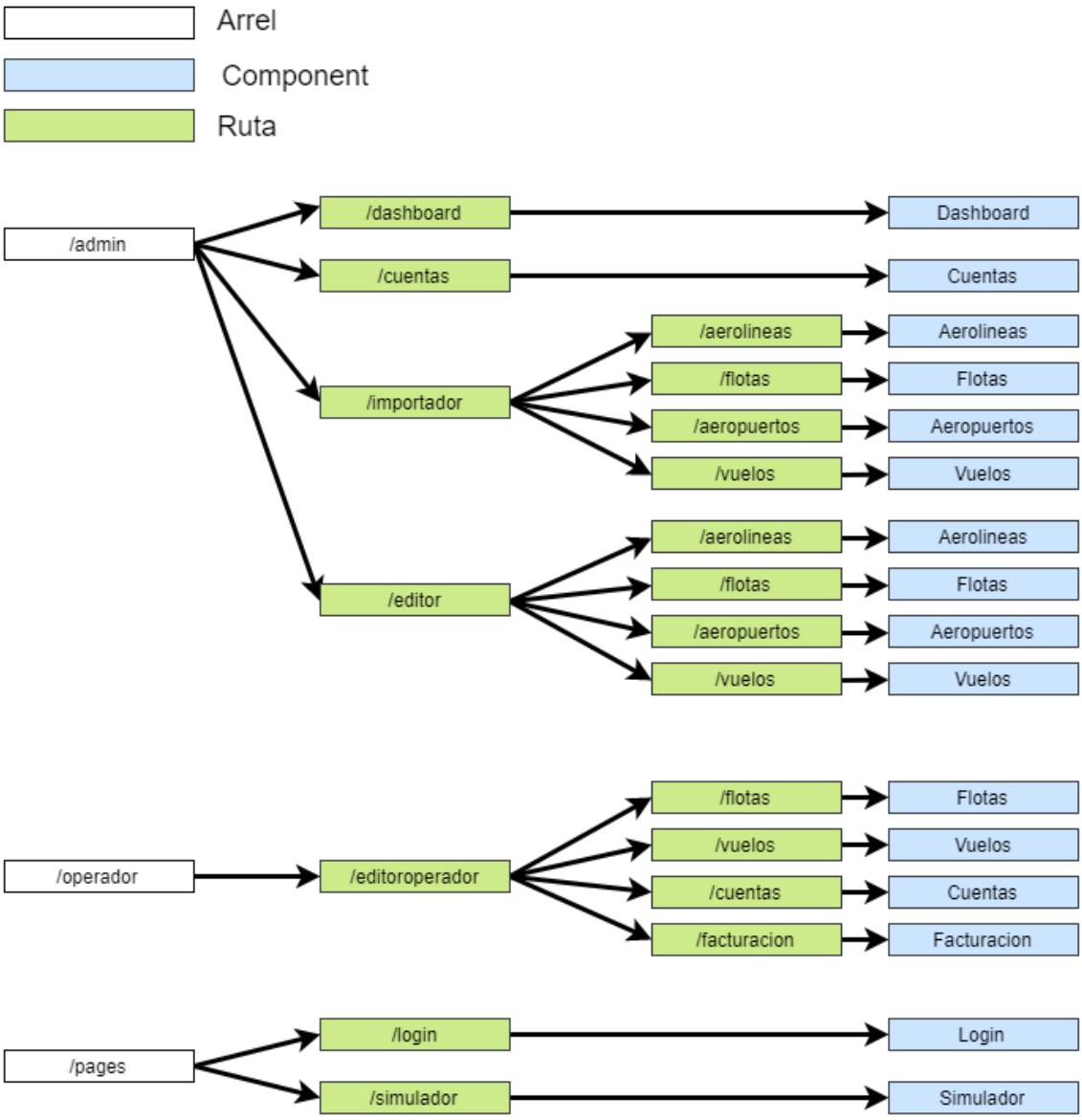


Fig. 4.23. Routing del projecte

4.4.4. Apollo Client

El client Apollo és la implementació de GraphQL en la part web. És l'encarregat de realitzar les peticions a l'API del projecte. La finalitat d'utilitzar aquest client és l'afinitat amb el servidor. Això és perquè és desenvolupat per la mateixa comunitat de desenvolupament. Així doncs, a l'utilitzar Angular 4 com *framework* del projecte, s'implementa la llibreria *apollo-angular*.

Pel que respecta al desenvolupament d'Apollo en la plataforma web del projecte, es necessita inicialitzar una instància del propi client Apollo. En la figura posterior es pot visualitzar el codi de com s'ha implementat:

```
//Creamos el network interface
const networkInterface = createNetworkInterface( uriOrInterfaceOpts: {
  uri: 'http://localhost:3000/graphql'
});

//Creamos el cliente
const client = new ApolloClient({
  networkInterface: networkInterface,
});

export function provideClient(): ApolloClient {
  return client;
}

@NgModule({
  imports: [
    BrowserModule,
    AppRoutingModule,
    BsDropdownModule.forRoot(),
    TabsModule.forRoot(),
    ApolloModule.forRoot(provideClient)
  ],
```

Fig. 4.24. Implementació del Client Apollo en *app.module.ts*

Tot el procés es realitza a dintre de l'*app.module.ts*, el mòdul arrel de la plataforma web. Inicialment es crea un objecte *networkInterface*. Aquest objecte és el connector de xarxa on s'especifica la direcció del servidor GraphQL. Per tant, és un objecte per configurar la part de xarxa del client Apollo.

A continuació es declara el client com una variable constant. Això permet exportar la funció *provideClient()* amb la finalitat de proporcionar un client Apollo configurat per tota la plataforma web. Per fer-ho, l'últim pas és importar de manera global el client Apollo utilitzant la sintaxi *ApolloModule.forRoot(provideClient).* D'aquesta manera hi haurà una instància del client en tots els mòduls i components del projecte. Això farà transformar el client Apollo en un servei d'Angular. Així doncs, un servei d'Angular té la finalitat d'actuar com un proveïdor de funcions o dades. Qualsevol mòdul del projecte té accés al servei declarant-lo a la definició del component (providers: [service]).

```
Aerolineas(query: string): Promise<any> {
  return new Promise((resolve) => {
    this.apollo.watchQuery( options: {
      query: gql(query),
      fetchPolicy: 'network-only',
    }).subscribe( next: ({data}) => {
      resolve(data);
    });
  });
};
```

Fig. 4.25. Implementació del servei *apollo-graphql.service.ts*

La figura anterior exposa la implementació del client Apollo dintre del servei d'Angular. Un dels avantatges de la utilització de GraphQL és demanar les dades que únicament siguin necessàries. Per tant, en la figura únicament es

defineix una funció amb un paràmetre 'query'. Aquest paràmetre inclou la *query* en un format string, que posteriorment, és traduïda a la sintaxis de GraphQL amb la funció *gpl()* en el camp d'opcions.

A part, una de les funcionalitats del client és mantindré un *cache* de les peticions per tal d'alliberar càrrega en el servidor. Contràriament, es configura amb una política de '*network—only'.* En altres paraules, totes les peticions han de ser enviades al servidor sense aplicar cap lectura del *cache*. Així s'evita un error comú, en el que no s'actualitza correctament les dades demanades, ja que el client, en la majoria de casos, entén com actualitzades les dades de la *cache.*

En conclusió, cada component d'Angular ha de cridar a la funció que necessiti del servei i incloure'n la *query* amb els camps que necessita.

4.5. Servidor de Lectura

El servidor de lectura és un element indispensable per realitzar l'objectiu de llegir els clauers RFID. La funcionalitat que realitza és bàsicament llegir l'identificador del clauer per subministrar la lectura a la plataforma web.

La forma d'implementar és bàsicament la creació d'un *script* amb el llenguatge Python. Inicialment s'importa la llibreria dedicada a la lectura dels connectors GPIO de la Raspberry Pi. Aquesta permet la connexió del mòdul MF-RC522 amb el propi codi del servidor. Per l'altra banda, hi ha la llibreria MFRC522 [33] dedicada a gestionar la lectura dels clauers amb protocol Mifare Classic 1k. Aquestes llibreries es troben importades al fitxer d'iniciació server.js del servidor.

En el conjunt d'ambdues, s'aconsegueix llegir l'identificador del clauer passat pel lector. A continuació, la figura amb el codi complet:

```
import RPi.GPIO as GPIO
import MFRC522

continue_reading = True

# Create an object of the class MFRC522
MIFAREReader = MFRC522.MFRC522()

# Welcome message

# This loop keeps checking for chips. If one is near it will get the UID and authenticate
while continue_reading:

    # Scan for cards
    (status,TagType) = MIFAREReader.MFRC522_Request(MIFAREReader.PICC_REQIDL)

    # If a card is found
    if status == MIFAREReader.MI_OK:

    # Get the UID of the card
        (status,uid) = MIFAREReader.MFRC522_Anticoll()

    # If we have the UID, continue
    if status == MIFAREReader.MI_OK:

        # Print UID
        print str(hex(uid[0])[2:])+":"+str(hex(uid[1])[2:])+":"+str(hex(uid[2])[2:])+":"+str(hex(uid[3])[2:])
        continue_reading = False

GPIO.cleanup()
```

Fig. 4.26. Script Python per la lectura dels clauers RFID

La implementació, a escala de servidor, parteix de desenvolupar un servidor Express bàsic amb una ruta de lectura (‘/lectura’). Aquesta ruta executarà la llibreria *python-shell* de la mateixa manera com s’ha utilitzat en els anteriors apartats. En cas de rebre una petició, el servidor espera fins que s’hagi llegit el clauer i enviarà el resultat de la petició. De forma similar, s’utilitza PythonShell per l’obtenció dels resultats aportats per al script. En resum, la implementació queda de la següent forma:

```
app.get('/lectura', (req, res) => {
  return new Promise((resolve, reject) => {
    PythonShell.run('Lectura.py',
      {
        mode: 'text',
        pythonOptions: ['-u'],
      }, (err, results) => {
        if (err) throw err;
  console.log(err);
        err ? reject(err) : resolve({idTag: results[0]});
      });

  }).then((data) =>{
    res.send(data);
  })
});
```

Fig. 4.27. Execució del script

4.6. App

En aquest apartat es vol exposar breument la implementació i el desenvolupament de l’aplicació mòbil. Aquesta té com a finalitat donar la lectura dels clauers RFID utilitzats en el projecte. També oferir la capacitat d’obtenir informació de l’estat de l’equipatge com de l’historial de l’usuari d’equipatges ja registrats anteriorment.

Un dels punts claus en el desenvolupament de l’App és la utilització del framework Ionic. Ionic [34] és un SDK de codi obert per el desenvolupament d’aplicacions utilitzant tecnologies web com CSS, HTML5 i Sass. Internament utilitza Typescrypt com a llenguatge de programació i Angular 4 com framework per el control dels components interns. Ionic també incorpora un conjunt de plugins natius anomenats Ionic Native. En altres paraules, Ionic té capacitat d’accedir al hardware del dispositiu mòbil. D’aquesta manera, en els dispositius compatibles, es pot llegir clauers RFID gràcies a l’accés de Ionic Native al controlador RFID/NFC dels telèfons.

Un dels avantatges en desenvolupar codi en el projecte és el reciclatge parcial del codi implementat en la plataforma web. Tant plataforma web, com l’aplicació mòbil utilitzen el mateix framework Angular i llenguatge Typescript. Així doncs, la implementació ha sigut més ràpida en comparació amb el desenvolupament del portal web. Per últim, en el següent diagrama es pot

observar els procediments i les vistes implementades per complir amb els objectius dissenyats per aquest projecte:

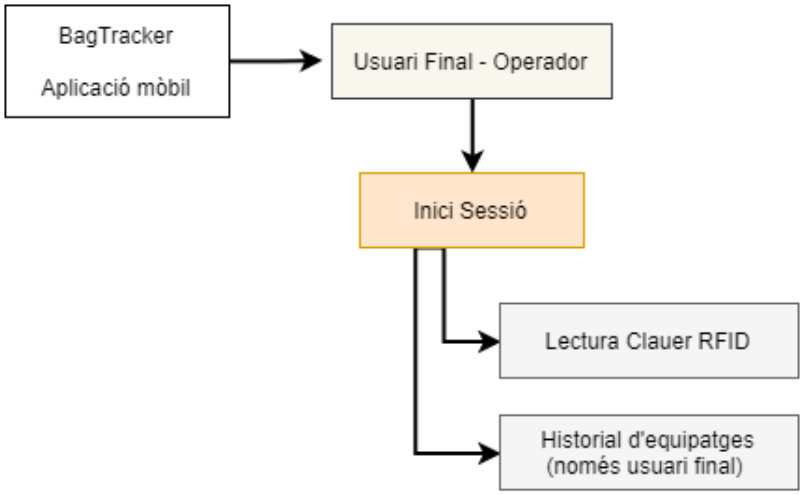


Fig. 4.28. Diagrama de vistes/procediments de l'aplicació mòbil

A continuació, s'exposen els dos procediments més sofisticats de l'aplicació mòbil. Pel que fa a les vistes funcionals, es poden trobar en l'apartat d'annexos del document.

4.6.1. Inici Sessió

Una de les característiques de l'aplicació és l'inici de sessió. La finalitat d'aquesta es la necessitat d'autenticar l'usuari i que l'aplicació pugui enviar peticions a l'API, evitant que siguin rebutjades per no incloure cap *token* (veure l'apartat de seguretat per més informació). Per tant, la primera imatge que veu l'usuari és una vista d'inici de sessió:

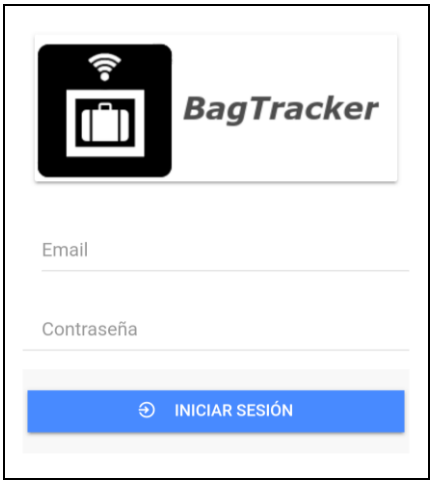


Fig. 4.29. Inici de sessió en l'aplicació mòbil

Pel que fa a la implementació, s'utilitza codi reciclat que forma part del component login de la plataforma web. Així doncs, l'aplicació mòbil emmagatzema les dades de l'inici de sessió en el *localstorage* del dispositiu.

```
login(): void {
  console.log(IP);
  this.http.post( url: IP +'/login',  body: {email: this.cuenta.email, pass: this.cuenta.pass}).subscribe( next: data => {
    console.log(data);
    let body = JSON.parse(data["_body"]);
    // Read the result field from the JSON response.
    if (body["log"] == 'Autenticado'){
      localStorage.setItem( key: 'usuario', JSON.stringify( value: {usuario: body["usuario"], token: body["token"]}));
      this.navCtrl.push(TabsPage);
    }
    else{
      alert("Usuario o contraseña no correctos");
    }
  });
```

Fig. 4.30. Implementació de l'inici de sessió

Un cop realitzada l'autentificació, el mateix sistema redirigeix cap a una nova vista general amb una barra inferior de navegació. Segons el rol de l'usuari, es mostraran totes les opcions o únicament la vista del lector dels clauers RFID.

4.6.2. Lectura clauer RFID

La lectura del clauer RFID és pot resumir en tres part diferenciades: la iniciació/lectura, recuperació, i l'obtenció de dades del servidor:

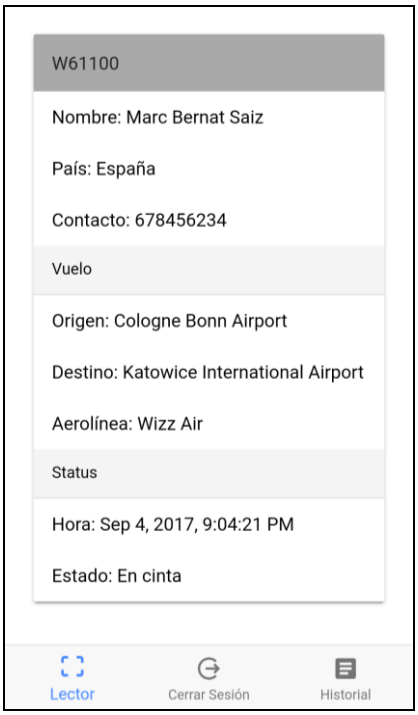


Fig. 4.31. Lectura realitzada d'un clauer RFID

En un primer principi, l'aplicació obté del *localstorage* el token necessari per realitzar peticions al servidor GraphQL. Sense el token, com en el cas de la plataforma web, l'API rebutjarà qualsevol petició. A continuació és crida la funció plataform.ready(). Aquesta retorna una premissa que indica que tant Ionic com Ionic Native han sigut carregats satisfactòriament i poden ser utilitzats. Així doncs, un cop el sistema carregat, el mòdul *androidPermissions* avaluà si el dispositiu té els permisos per utilitzar el NFC integrat en el telèfon mòbil. En cas contrari, demanarà permisos amb una finestra d'alerta. Per últim, s'activa el mòdul NFC d'Ionic Native, en mode espera, per llegir qualsevol clauer RFID. I en cas de detectar algun, en crea un esdeveniment/*event* amb la informació llegida i l'envia a la funció 'tagListenerSuccess()'.

```
ngOnInit () {
  let user = localStorage.getItem( key: 'usuario');
  console.log(JSON.stringify(user["usuario"]));
  user = JSON.parse(user);
  user = ["usuario"]["operador"];
  console.log(JSON.stringify(user));
  this.platform.ready().then( onfulfilled: () => {
    this.androidPermissions.checkPermission(this.androidPermissions.PERMISSION.NFC).then(
      onfulfilled: success => console.log('Permission granted'),
      onrejected: err => this.androidPermissions.requestPermissions(this.androidPermissions.PERMISSION.NFC)
    );
    this.nfc.addTagDiscoveredListener().subscribe( next: (tagEvent:Event) => this.tagListenerSuccess(tagEvent));
  });
```

Fig. 4.32. Inicialització/lectura del lector NFC

En segon lloc, s'executa la transformació de l'esdeveniment anterior per extreure l'identificador del clauer. Aquest està codificat, en una base binaria, que posteriorment es transforma en caràcters de codificació hexadecimal. Així doncs, es separa la cadena de caràcters resultant en formacions de dues unitats separades per dos punts (:). L'objectiu d'aquest procés és la d'igualar l'identificador del clauer llegit amb el format que existeix en la plataforma web i el servidor.

```
tagListenerSuccess(tagEvent) {
  console.log("tag:" + JSON.stringify(tagEvent));
  this.tag = this.chunk(this.nfc.bytesToHexString(tagEvent.tag.id),  n: 2).join(':');
  this.obtenerinfoTag();

}
```

Fig. 4.33. Transformació de l'esdeveniment

Finalment, s'executa la funció obtenerinfoTag(). Aquesta és l'encarregada d'agafar l'identificador i realitzar la petició a l'API per obtenir les dades de l'equipatge que es mostraran a l'usuari.

```
obtenerinfoTag(): void {
  this.service.equipajeidTag(this.tag).then( onfulfilled: (data) => {
    this.maleta = data.equipaje_idtag;
    console.log(JSON.stringify(this.maleta));
    this.ultimo = this.maleta.status[this.maleta.status.length -1];
    console.log(this.ultimo);
  })
```

Fig. 4.34. Obtenció dades del servidor

4.7. Seguretat

En aquest apartat es vol exposar els mecanismes que s'utilitzen en la comunicació entre el servidor web, l'aplicació mòbil i la plataforma web. La finalitat d'implementar seguretat en el projecte és la d'evitar l'ús inadequat de les funcionalitats presents per persones no autoritzades. En resum, hi existeixen dos mecanismes de seguretat: un a nivell de peticions i l'altre a nivell de *routing.* Excloent la pròpia autentificació tant de l'aplicació mòbil com la de la web.

4.7.1. JWT

JWT [35] és un estàndard obert, basat en JSON, per crear tokens d'accés que permeten l'ús de recursos d'una aplicació o d'una API. Aquest token porta la informació de l'usuari que necessita el servidor per identificar-ho, així com informació addicional que pugui ser-li útil (rols, permisos, etc.).

EL JWT incorpora dos mecanismes de seguretat. El primer és la incorporació d'un temps de validesa. Quan el valor en temps expira, el token deixa de ser vàlid. El segon és l'ús d'un algoritme de verificació per realitzar una firma digital. Així doncs, es permet verificar que el generador del token és el servidor i que el propi token no ha sigut modificat. En el cas del servidor, s'utilitza una clau secreta (clau simètrica) per firmar els token.

```
export function login (req, res){
  mongo.Usuarios.findOne({email: req.body.email}, (err, usuario) => {
    if (err) reject(err);

    if (!usuario || !usuario.comparePassword(req.body.pass)) {
      res.send({ token: '', log: 'Autentificación fallida. Usuario o contraseña no válido' });
    }
    else {
      res.send({
        token: jwt.sign({
          sub: usuario._id,
          iat: moment().unix(),
          exp: moment().add(1, "hours").unix(),
          name: usuario.nombre,
          email: usuario.email
        }, jwtkey),
        log: "Autenticado",
        usuario:{
          nombre: usuario.nombre,
          admin: usuario.admin,
          operador: usuario.operador,
          id: usuario._id,
          idoperador: usuario.idoperador || '',
        }});
    }

  });
};
```

Fig. 4.35. Implementació JWT amb clau simètrica

En la figura superior es pot observar com inicialment es realitza una comprovació per verificar si la contrasenya coincideix amb el *hash* del *password*. En cas afirmatiu, el servidor emet un token JWT. En la implementació del servidor, a part del token, s'envia informació extra com el propi model de l'usuari i un camp log. La finalitat d'incloure més informació és ajudar a diferenciar el tipus d'usuari, administrador o operador, en l'aplicació web o l'app mòbil. Com a extra, el token té una durada d'una hora, en altres paraules, el token generat té una duració d'uns 60 minuts.

Contràriament, hi ha un escenari que pot fer dubtar l'eficiència de la implementació. És el cas d'un usuari que pugui modificar el token i habilitar l'opció d'administrador o operador. Aquí és on entra l'ús de capçaleres per part del client Apollo i l'*endpoint* del servidor.

```
networkInterface.use( middlewares: [{
  applyMiddleware(req, next) {
    if (!req.options.headers) {
      req.options.headers = {};  // Create the header object if needed.
    }
    // get the authentication token from local storage if it exists
    let local = localStorage.getItem( key: 'usuario');
    local = JSON.parse(local);
    req.options.headers.authorization = local["token"] ? `Bearer ${local["token"]}` : null;
    next();
  }
}]);
```

Fig. 4.36. Implementació capçalera en peticions de GraphQL

Qualsevol petició d'obtenció o modificació de dades passa per el servidor. En primer lloc, comprova si existeix el camp *headers* en la petició HTTP. En cas afirmatiu, recupera el token emmagatzemat del *localstorage* del navegador i l'insereix dintre de la capçalera 'authoritzation'.

Quan arriba al servidor aquest executa la funció *.isAuth().* Aquesta funció verifica, en primer terme, si existeix la capçalera 'authoritzation' en la petició rebuda. Si és afirmatiu, realitza la comprovació de la firma. En tots els casos, el servidor és l'únic que sap la clau simètrica, i per tant, pot detectar si hi ha hagut un intent de manipular el token.

En resum, si la funció *isAuth()* denega la validesa del token aportat per la petició, el servidor bloqueja qualsevol incentiva d'obtenir o modificar dades del sistema.

```
export function isAuth (req, res, next) {
  if (!req.headers.authorization) {
    return res
      .status(403)
      .send({message: "La petición no tiene cabecera de autorización"});
  }

  let token = req.headers.authorization.split(" ")[1];
  console.log(token);
  try {
    jwt.verify(token, jwtkey, (err, decoded) => {

      if (err){
        return res
          .status(401)
          .send({message: "El token no es válido"});
      }
      else{
        next();
      }
    });
  } catch (err){
    console.log(err);
    return res
      .status(401)
      .send({message: "El token ha expirado"});
  }

}
```

Fig. 4.37. Comprovació de la firma del token

4.7.2. Angular Guard

Angular Guard [36] és el nom que rep la implementació de protegir les rutes d'Angular amb qualsevol mecanisme de seguretat. La forma més senzilla és implementar un servei que inclogui els *Navigation Guards* del router d'Angular.

Els *Navigation Guard* estan formats per quatre tipus segons l'objectiu:

- CanActivate: Decideix si una ruta pot ser activada.
- CanActivateChild: Decideix si rutes fills / children poden ser activades.
- CanDeactivate: Decideix si una ruta pot ser desactivada.
- CanLoad: Decideix si un mòdul pot ser carregat.

Així doncs, en el projecte únicament utilitzarem les funcions CanActivate i CanActivateChild. La funció CanActivate s'utilitza per protegir les rutes arrel, /admin i /operador, i per evitar l'entrada de personal no autoritzat en la plataforma web. En el cas de la funció CanActivateChild s'utilitza com funció temporitzador. Aquesta comprova si el token no ha expirat en temps. En cas afirmatiu (expiració del token), retorna a l'usuari a la pàgina d'iniciar sessió.

```
canActivate(): boolean {
  try {
    let check = localStorage.getItem( key: 'usuario');
    check = JSON.parse(check);
    if (localStorage.getItem( key: 'usuario') && check["usuario"]["operador"]) {
      // logged in so return true
      return true;
    }
    // not logged in so redirect to login page
    this.router.navigate( commands: ['pages/login']);
  }
  catch (e){
    // not logged in so redirect to login page
    this.router.navigate( commands: ['pages/login']);
  }
}

canActivateChild() {

  let jwtHelper: JwtHelper = new JwtHelper();

  let check = localStorage.getItem( key: 'usuario');
  check = JSON.parse(check);
  console.log(check);
  //this.jwhelper se utiliza para comprobar si esta antiguo;
  try {
    if (jwtHelper.isTokenExpired( token: check["token"] || '') || !check){
      alert("La sessión ha expirado");
      this.router.navigate( commands: ['pages/login']);
      return false;
    }
    else{
      return true;
    }
  }catch (e){
    console.log(e);
    alert("La sessión ha expirado");
    this.router.navigate( commands: ['pages/login']);
  }


}
```

Fig. 4.38. Implementació d'Angular Guards de l'operador

Per concloure, la manera d'implementar el servei és afegint un camp amb el nom del *Navigation Guard* en el fitxer *app.routing.ts* del projecte. Després, amb uns claudàtors s'inclou el mòdul servei que implementa les funcions. En la figura següent, s'exposa el mòdul AuthGuard per les rutes de l'administrador.

```
{
  path: 'admin',
  component: FullLayoutComponent,
  canActivate: [AuthGuard],
  data: {
    title: 'Home'
  },
  children: [
    {
      canActivateChild: [AuthGuard],
      path: 'dashboard',
      loadChildren: './dashboard/dashboard.module#DashboardModule'
    },
    {
      canActivateChild: [AuthGuard],
      path: 'importador',
      loadChildren: './importador/importador.module#ImportadorModule'
    },
    {
      canActivateChild: [AuthGuard],
      path: 'editor',
      loadChildren: './editor/editor.module#EditorModule'
    },
    {
      canActivateChild: [AuthGuard],
      path: 'cuentas',
      loadChildren: './cuentas/cuentas.module#CuentasModule'
    }
  ]
},
```

Fig. 4.39. Implementació d'Angular Guards en *app.routing.ts*

4.8. Entorn de desenvolupament

En quant a l'entorn de desenvolupament per realitzar les diferents parts del projecte s'han utilitzat les següents eines:

4.8.1. WebStorm

Webstorm [37] és un IDE comercial per Web, Javascript, Typescript desenvolupat per JetBrains. Està especialitzat en desenvolupaments de clients (*front-end/client-side)* i servidors (server-side) amb Node.js. Compta amb funcionalitats com l'anàlisi intel·ligent d'errors, refactorització, autocompletar sintaxis, control de versions, etcètera. Al ser un producte comercial s'ha utilitzat la llicència per estudiants universitaris.

Pel que fa a la implementació del projecte: plataforma web, servidors i aplicació mòbil; s'ha utilitzat l'IDE Webstorm.

4.8.2. Git

Git [38] és un sistema de control de versions dissenyat per Linus Torvalds, pensat en l'eficiència i el manteniment de versions d'aplicacions amb una enorme quantitat de fitxers de codi font. Com a gestor s'ha utilitzat per interfície de línia de comandes.

Git s'ha utilitzat per el control de versions d'algunes parts del projecte com l'importador pyfligthdata o la implementació del lector extern de clauers RFID.

4.8.3. Spyder

Spyder [39] és un entorn integrat multi plataforma de desenvolupament pel llenguatge de programació Python. Conté un conjunt de *plugins* integrats per la inspecció de dades, eines de construcció automàtiques o revisió intel·ligent de la sintaxis.

Spyder s'ha utilitzat com a editor dels codis escrits en Python com els scripts utilitzats en l'importador pyflightdata.

4.8.4. NPM

NPM [40] és un gestor de paquets per al llenguatge de programació JavaScript. És el gestor de paquets predeterminat de l'entorn d'execució de Node.js. Consisteix en un client de línia d'ordres i una base de dades en línia de paquets públics i de pagament. El registre es pot accedir a través del client, i els paquets disponibles es poden explorar i cercar a través del lloc web de npm.

En el projecte és l'encarregat de descarregar i instal·lar els paquets indicats en el fitxer package.json de cada element del projecte. En el cas del servidor, és l'encarregat de mantenir els paquets instal·lats i gestionar la iniciació del servidor amb la comanda 'npm start'.

4.8.5. Robomongo

Robomongo és un editor gràfic multi plataforma per l'administració de les bases de dades MongoDB.

En el projecte s'ha utilitzat per realitzar comprovacions i modificacions en els models de dades.

CAPÍTOL 5. PLANIFICACIÓ

En el present capítol es vol exposar la planificació realitzada pel compliment dels objectius marcats en el projecte.

- Estudi previ: Aquesta tasca engloba tot el període de recerca i estudi del context del projecte. Estudi de les necessitats actuals. Definició dels objectius que es volen portar a terme.
- Especificacions: En aquest grup es defineixen totes les tasques de: plantejament de les funcionalitats, disseny dels models de dades, hardware a utilitzar.
- Estudi tècnic: Correspon a l'elecció de totes les tecnologies necessàries per implementació del projecte. Anàlisi de l'estat de l'art actual.
- Implementació: Aquí s'engloben totes les tasques de programació, desenvolupament i de disseny visual.
- Test: Comprovació i reparació d'errors en durant la implementació i el desenvolupament del projecte.
- Documentació: Redacció de la memòria del treball.

	Concepte		Inici	Final
Estudi Previ				
	Recerca del context		Febrer de 2017	Febrer de 2017
	Objectius del projecte		Febrer de 2017	Febrer de 2017
Especificacions				
	Plantejament de les funcionalitats		Febrer de 2017	Març de 2017
	Elecció de les tecnologies		Març de 2017	Març de 2017
	Disseny del model de dades i elecció Hardware		Març de 2017	Març de 2017
Estudi Tècnic				
	Cursos i projectes de pràctiques		Abril de 2017	Abril de 2017
Implementació				
	Importadors		Maig de 2017	Maig de 2017
	Servidor		Maig de 2017	Juliol de 2017
	Lector/Raspberry Pi		Maig de 2017	Maig de 2017
	Client		Maig de 2017	Juliol de 2017
	Aplicació mòbil		Juliol de 2017	Agost de 2017
Test				
	Global		Maig de 2017	Agost de 2017
Documentació				
	Realització de la memòria escrita		Agost de 2017	Agost de 2017

Taula 5.1. Planificació temporal del projecte

Tal com mostra la taula, el primer mes ha sigut el més lleuger en càrrega lectiva. S'ha basat en la confecció de la idea del projecte i la definició dels objectius. L'estudi previ ha valgut per valorar les funcionalitats assolibles per la creació del projecte.

En un segon terme, es planteja les funcionalitats per cobrir les necessitats dels objectius. A continuació, s'han ideat els models de dades. A diferència d'altres projectes anteriors, s'ha volgut donar importància a crear un model de dades que utilitzes els avantatges de les tecnologies elegides. Per una altra banda, es contempla la utilització del model MF-RC522 com a suport del servidor per la lectura dels clauers RFID.

Posteriorment, la corba d'aprenentatge s'incrementa molt ràpidament degut a la necessitat d'aprenentatge de nous llenguatges de programació. En total s'estudien dos llenguatges de programació, la nova forma de desenvolupar Angular, partint de la base coneguda d'AngularJS que fa caure en conflictes per la seva similitud, i l'estudi de l'ecosistema GraphQL que compta amb molt poca documentació si es compara amb altres projectes.

Com a conseqüència de la necessitat de realitzar un estudi intensiu, s'elaboren projectes de pràctiques per tal de començar a assimilar els continguts de les tecnologies. Amb la finalitat de desenvolupar el projecte amb la màxima correcció d'errates a nivell estructurat o d'implementació.

Tot seguit comporta una disminució de la pendent de la corba d'aprenentatge a conseqüència de portar una preparació prèvia gràcies a l'estudi tècnic anterior. Així doncs, es comença a treballar en el projecte elaborant els primers importadors del sistema. Seguidament s'inicià el desenvolupament paral·lel de l'aplicació web i el servidor. En un termini curt, s'abandonen els anteriors per desenvolupar el Lector dels clauers amb la Raspberry Pi. En finalitzar, es continua amb el desenvolupament paral·lel.

Pel que fa a l'apartat de test, es realitzen tests durant les diverses etapes del projecte verificant el correcte funcionament del projecte. En la finalització del desenvolupament es realitza novament una inspecció d'errors.

Finalment el projecte es materialitza amb la realització de la memòria escrita del treball.

CAPÍTOL 6. CONCLUSIONS

D'aquest treball es desprèn la necessitat d'aportar una solució per tal de minimitzar la problemàtica actual en la logística del sector aeronàutic. L'avanç de les tecnologies permet oferir noves solucions a l'hora d'etiquetar de l'equipatge. No obstant això, actualment encara hi ha deficiències, no solament, pels sistemes ja instal·lats. Realment recau en la poca iniciativa de canviar les coses amb l'objectiu de maximitzar el benefici a costa de l'experiència dels usuaris.

Un dels punts atractius per endinsar-se en el desenvolupament del treball, ha sigut la combinació de tecnologies ja comprovades amb les tecnologies emergents. N'és una mostra, la utilització de clauers RFID presents en el mercat farà un parell d'anys, mentre que en l'àmbit de codi, la versió d'Angular 2 va alliberar-se el setembre de 2016 (menys d'un any), la versió d'Angular 4 el març de 2017 (menys de 5 mesos) o GraphQL el març de 2016.

Encara que un dels propòsits ha sigut l'aprenentatge de nous llenguatges de programació per assolir la finalització dels objectius. El repte d'afrontar la gran quantitat de dades a transportar, com la forma de gestionar-les, ha sigut la problemàtica de cada hora de desenvolupament. No obstant, m'ha ajudat en l'adquisició de noves metodologies de treball i la millora en l'autoaprenentatge autònom. El fet de realitzar una planificació prèvia ha millorat el ritme de desenvolupament fent progressions de forma gradual.

En termes generals, s'han complert tots els objectius que s'havien presentat de forma satisfactòria. Pel que fa al projecte, s'ha creat un prototip o base per començar a desplegar nous projectes que incloguin un model de negoci associat. No obstant, té totes les mínimes funcionalitats per realitzar una prova pilot en un entorn real.

6.1. Futures millores

Amb la finalització del projecte hi ha un conjunt d'aspectes tècnics i funcionals que es podrien realitzar per ampliar l'actual treball:

En primer lloc, la creació de la xarxa de sensors tal com s'ha comentat en capítols anteriors. En altres paraules, la implementació d'un ecosistema basat en sensors que automàticament puguin realitzar modificacions dels estats de l'equipatge.

En segon lloc, l'ampliació del model de dades i del sistema. El projecte encara que és funcional i compleix els objectius, necessita una gran quantitat d'hores per perfeccionar la gestió d'errors, ampliar el projecte en la part de *Bussiness Logic, o* ampliar el nombre de funcionalitats presents. Caldria incloure informació de caràcter logístic, orientat als tecnicismes i/o en l'àmbit operatiu intern dels aeroports.

6.2. Impacte mediambiental

Una de les millores que destaquen del projecte és la utilització de clauers RFID. La utilització d'aquests és la reducció global en consum de paper, tintes químiques per la impressió d'etiquetes i la utilització de polímers plàstics.

Un dels principis bàsics del clauer és la seva reutilització. La idea principal és la recollida dels clauers un cop l'usuari ha finalitzat el seu itinerari. D'aquesta manera s'evita la fabricació massiva de clauers amb un únic ús.

També una idea futura és la substitució de les etiquetes de les maletes pels mateixos clauers RFID. D'aquesta manera es reduiria el consum de paper i de la impressió d'etiquetes.

A un nivell inferior, la millora en la logística aeronàutica pot suposar una reducció dels costs operatius. Es pot traduir en una disminució de moviments per part de tot el sector logístic a fi de reduir les emissions contaminants i una millora energètica.

BIBLIOGRAFIA

[1] Article: La odisea de las líneas aéreas para hallar maletas perdidas. (en línia). [Última consulta: 9 d' Agost de 2017]. URL: http://www.bbc.com/mundo/noticias/2015/04/150406_perdidas_maletas_viajes_aereos_mes

[2] Informe: Reporte de Equipaje. (en línia). [Última consulta: 9 d' Agost de 2017]. URL: https://es.sita.aero/_onelink_/sita/sites/www-sita-aero/en2es/pdf/sita-baggage-report-2016-spanish.pdf

[3] Article: Las maletas extraviadas empiezan a acumularse en El Prat. (en línia). [Última consulta: 9 d'Agost de 2017]. URL: http://www.elperiodico.com/es/economia/20160701/las-maletas-empiezan-a-acumularse-en-el-prat-5241695

[4] Article: Comunicació de camp proper. (en línia). [Última consulta: 100 d'Agost de 2017]. URL: https://ca.wikipedia.org/wiki/Comunicaci%C3%B3_de_camp_proper

[5] FAQ: What is a Raspberry Pi?. (en línia). [Última consulta: 10 d'Agost de 2017]. URL: https://www.raspberrypi.org/help/faqs/#introWhatIs

[6] Article: Raspberry Pi. (en línia). [Última consulta: 10 d'Agost de 2017]. URL: https://ca.wikipedia.org/wiki/Raspberry_Pi

[7] Fulla d'especificacions: MFRC522. (en línia). [Última consulta: 10 d'Agost de 2017]. URL: https://www.nxp.com/docs/en/data-sheet/MFRC522.pdf

[8] Fulla d'especificacions: MIFARE Classic EV. (en línia). [Última consulta: 10 d'Agost de 2017]. URL: http://www.nxp.com/docs/en/fact-sheet/MIFARE-CASSIC-EV1-FS.pdf

[9] Document: Sub Docs. (en línia). [Última consulta: 11 d'Agost de 2017]. URL: http://mongoosejs.com/docs/subdocs.html

[10] Article: WebAPI. (en línia). [Última consulta: 13 d'Agost de 2017]. URL: https://es.wikipedia.org/wiki/Web_API

[11] Article: API REST: qué es y cuáles son sus ventajas en el desarrollo de proyectos. (en línia). [Última consulta: 13 d'Agost de 2017]. URL: https://bbvaopen4u.com/es/actualidad/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos

[12] Article: Aprende ECMAScript 6 (ES6 o ES2015), el nuevo estándar de JavaScript. (en línia). [Última consulta: 13 d'Agost de 2017]. URL: https://carlosazaustre.es/ecmascript-6-el-nuevo-estandar-de-javascript/

[13] Web: Babel is a JavaScript compiler. (en línia). [Última consulta: 13 d'Agost de 2017]. URL: https://babeljs.io/

[14] Web: Node.js® es un entorno de ejecución para JavaScript. (en línia). [Última consulta: 13 d'Agost de 2017]. URL: https://nodejs.org/es/

[15] Web: Infraestructura web rápida, minimalista y flexible para Node.js. (en línia). [Última consulta: 13 d'Agost de 2017]. URL: http://expressjs.com/es/

[16] Article: MongoDB. (en línia). [Última consulta: 13 d'Agost de 2017]. URL: https://es.wikipedia.org/wiki/MongoDB

[17] Web: Mongoose. (en línia). [Última consulta: 13 d'Agost de 2017]. URL: http://mongoosejs.com/

[18] Article: GraphQL. (en línia). [Última consulta: 13 d'Agost de 2017]. URL: https://en.wikipedia.org/wiki/GraphQL

[19] Article: ¿Por qué deberíamos abandonar REST y empezar a usar GraphQL en nuestras APIs?. (en línia). [Última consulta: 13 d'Agost de 2017]. URL:https://www.genbetadev.com/desarrollo-aplicaciones-moviles/por-que-deberiamos-abandonar-rest-y-empezar-a-usar-graphql-en-nuestras-apis

[20] Article: Conceptos: Observables, RxJS y Angular 2. Javascript Reactivo y Funcional. (en línia). [Última consulta: 13 d'Agost de 2017]. URL: https://codekstudio.com/post-blog/conceptos-observables-rxjs-y-angular-2-javascript-reactivo-y-funcional/57d1e2840897131b5ec54b90

[21] Repositori: A reference implementation of GraphQL for JavaScript. (en línia). [Última consulta: 13 d'Agost de 2017] URL: https://github.com/graphql/graphql-js/

[22] Documentació: Apollo Tools Guide. (en línia). [Última consulta: 14 d'Agost de 2017]. URL: http://dev.apollodata.com/tools/

[23] Article: GraphQL Schema Language Cheat Sheet. (en línia). [Última consulta: 14 d'Agost de 2017] URL: https://wehavefaces.net/graphql-shorthand-notation-cheatsheet-17cd715861b6

[24] Article: Web Scraping. (en línia). [Última consulta: 14 d'Agost de 2017] URL: https://ca.wikipedia.org/wiki/Web_scraping

[25] Repositori. Pyflightdata. (en línia). [Última consulta: 15 d'Agost de 2017] URL: https://github.com/MarcBernat/pyflightdata

[26] Article: BeatifulSoup. (en línia). [Última consulta: 15 d'Agost de 2017] URL: https://es.wikipedia.org/wiki/Beautiful_Soup

[27] Repositori: python-sehll. (en línia). [Última consulta: 15 d'Agost de 2017]. URL: https://github.com/extrabacon/python-shell

[28] Article: Aplicacions de pàgina única. (en línia). [Última consulta: 15 d'Agost de 2017]. URL: https://ca.wikipedia.org/wiki/Aplicacions_de_p%C3%A0gina_%C3%BAnica

[29] Article: Bootstrap. (en línia). [Última consulta: 15 d'Agost de 2017]. URL: https://es.wikipedia.org/wiki/Bootstrap_(framework)

[30] Repositori: CoreUI - Free Bootstrap Admin Template. (en línia) [Última consulta: 15 d'Agost de 2017]. URL: https://github.com/mrholek/CoreUI-Free-Bootstrap-Admin-Template

[31] Article: TypeScript. (en línia). [Última consulta: 15 d'Agost de 2017]. URL: https://es.wikipedia.org/wiki/TypeScript

[32] Article: Angular. (en línia). [Última consulta: 16 d'Agost de 2017]. URL: https://es.wikipedia.org/wiki/Angular_(framework)

[33] Repositori: MFRC522-python. (en línia). [Última consulta: 17 d'Agost de 2017]. URL: https://github.com/mxgxw/MFRC522-python

[34] Article: Ionic. (en línia). [Última consulta: 17 d'Agost de 2017]. URL: https://en.wikipedia.org/wiki/Ionic_(mobile_app_framework)

[35] Article: Refresh token con autenticación JWT. Implementación en Node.js. (en línia). [Última consulta: 17 d'Agost de 2017]. URL: https://solidgeargroup.com/refresh-token-autenticacion-jwt-implementacion-nodejs?lang=es

[36] Article: Protecting routes using guards in angular. (en línia). [Última consulta: 18 d'Agost de 2017]. URL: https://blog.thoughtram.io/angular/2016/07/18/guards-in-angular-2.html

[37] Article: WebStorm - The Smartest JavaScript IDE. (en línia). [Última consulta: 19 d'Agost de 2017]. URL: https://confluence.jetbrains.com/display/WI/WebStorm+IDE

[38] Article: Git. (en línia). [Última consulta: 19 d'Agost de 2017]. URL: https://ca.wikipedia.org/wiki/Git

[39] Article: Syper. IDE. (en línia). [Última consulta: 19 d'Agost de 2017]. URL: https://en.wikipedia.org/wiki/Spyder_(software)

[40] Article: NPM. (en línia). [Última consulta: 19 d'Agost de 2017]. URL: https://en.wikipedia.org/wiki/Npm_(software)

ACRÒNIMS

API: Application **P**rogramming **I**nterface. És un conjunt d'especificacions per a la comunicació entre diferents components programari.

RFID: Radio **F**requency **I**dentification

QR: Quick **R**esponse code

NFC: Near **F**ield **C**ommunication

SBC: Single **B**oard **C**omputer

PCB: Printed **C**ircuit **B**oard

GPIO: General **P**urpose **I**nput**/O**utput.

ISO: International **O**rganization for **S**tandardization com sigla d'estàndard.

IATA: International **A**ir **T**ransport **A**ssociation.

ICAO: **I**nternational **C**ivil **A**viation **O**rganization.

URI: Universal **R**esource **I**dentifier

SPA: Single **P**age **A**pplication

HTTP: Hyper**t**ext **T**ransfer **P**rotocol

REST: Representational **S**tate **T**ransfer

JSON: Java**S**cript **O**bject **N**otation

ODM: Object **D**ocument **M**apper

HATEOAS: Hypermedia **a**s **t**he **E**ngine **o**f **A**pplication **S**tate

SDK: Software **D**evelopment **K**it

IDE: Integrated **D**evelopment **E**nviroment

Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

ANNEXOS

TÍTOL DEL TFG: Plataforma per la gestió d'equipatges del sector aeroportuari

TITULACIÓ: Grau en Enginyeria Telemàtica

AUTOR: Marc Bernat Saiz

DIRECTOR: Toni Oller Arcas

DATA: 4 de setembre del 2017

ANNEXOS

1.1. GraphiQL

GraphiQL és una interfície visual inclosa en la implementació de GraphQL. Bàsicament és una eina que facilita la creació de consultes, la correcció d'errors en *querys* i *mutations* (peticions), i exposa totes les peticions disponibles en el servei GraphQL en forma de documentació.

En els exemples a continuació es pot veure l'autocompletat de camps i la correcció d'errors. Per últim, una visió general de l'eina amb els tres sectors diferenciats: la caixa de text, el visualitzador de respostes JSON i el requadre de documentació.

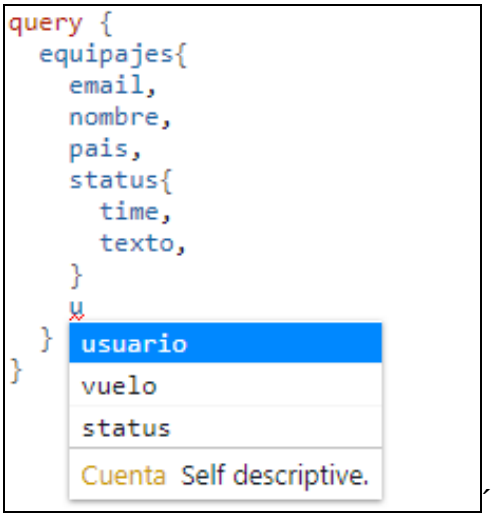


Fig. 1.1. Autocompletat de camps

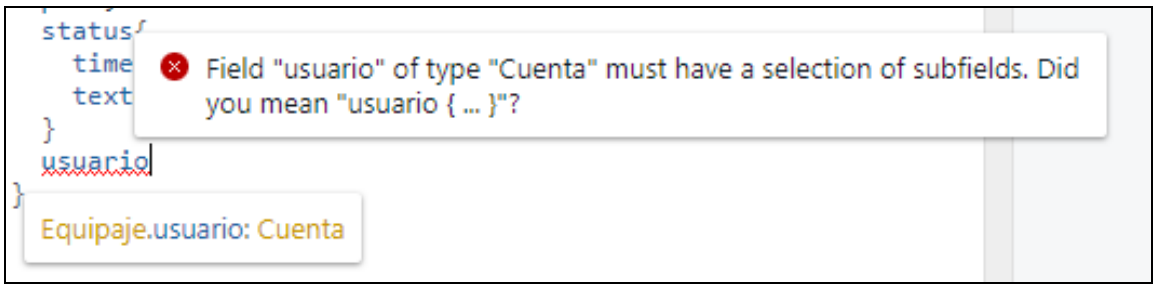


Fig. 1.2. Avís d'errors

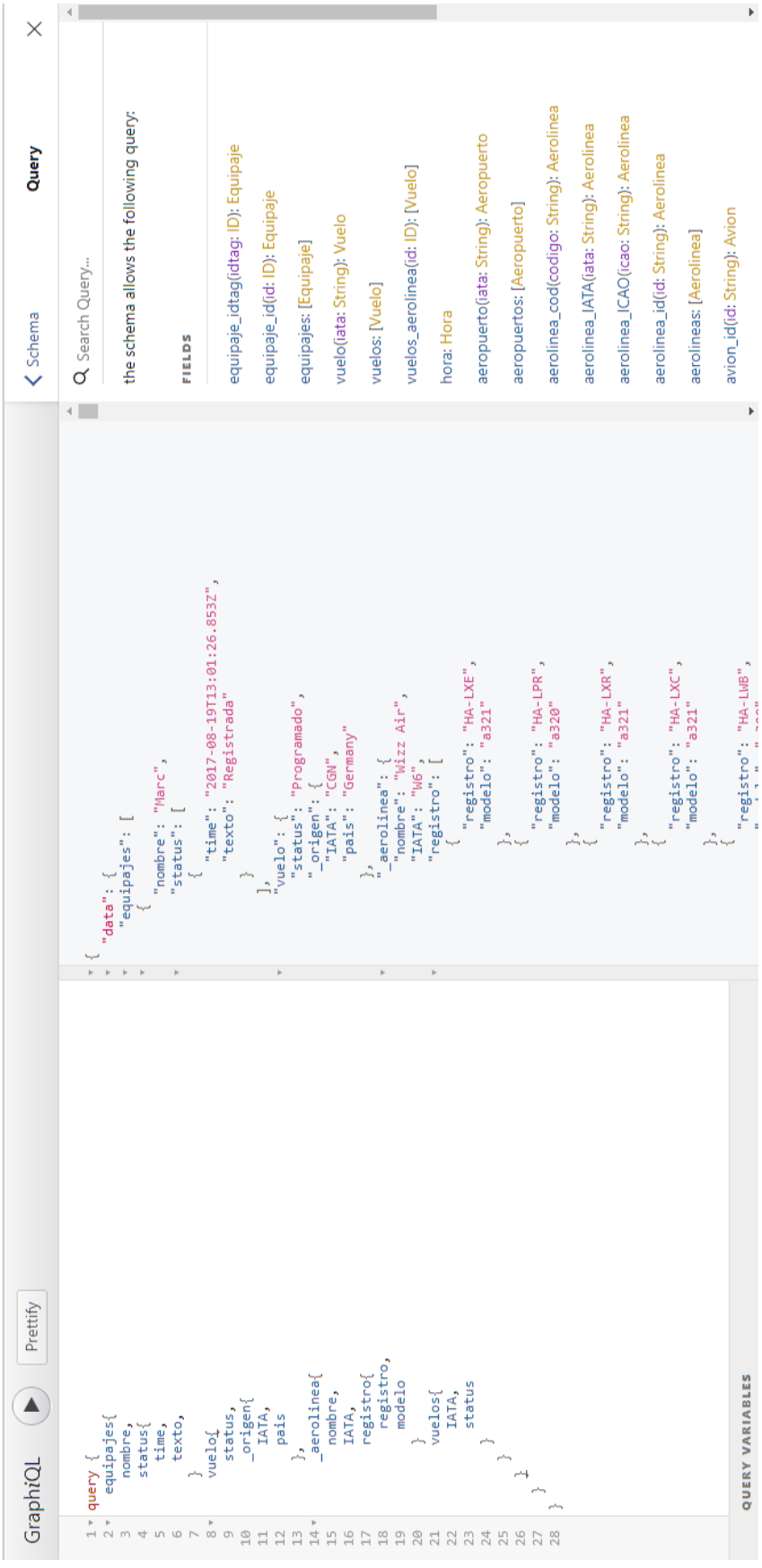


Fig. 1.3. Vista general de GraphiQL

1.2. Disseny Plataforma Web

Inici de sessió general per els usuaris administradors i operadors:

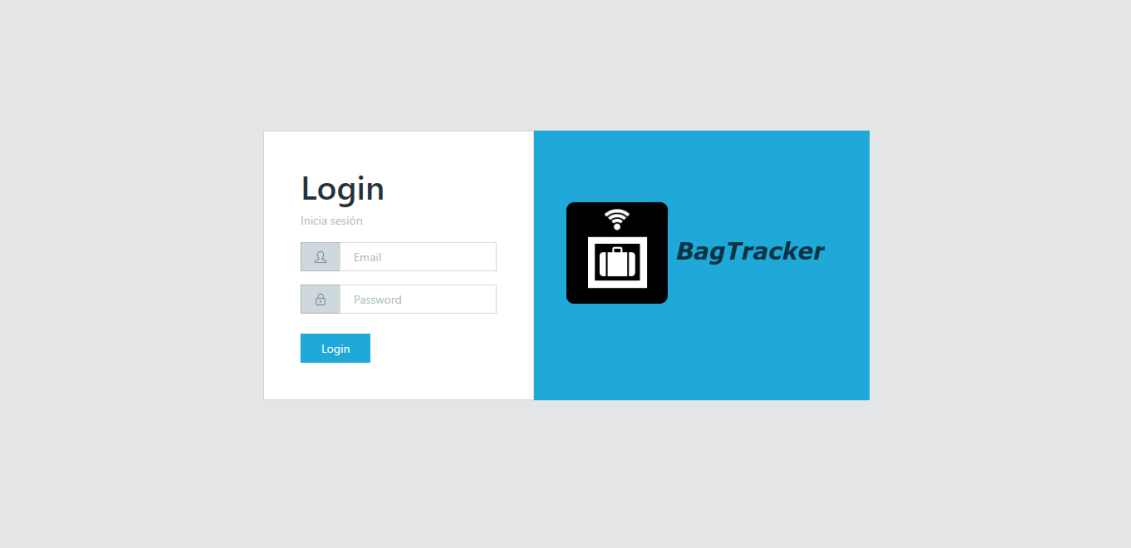


Fig. 2.1. Vista inici de sessió

1.2.1. Administrador

A continuació s'exposen les vistes de la plataforma web amb rol d'administrador:

1.2.1.1. Vista de l'administrador

Pàgina principal de BagTracker en la versió d'administrador

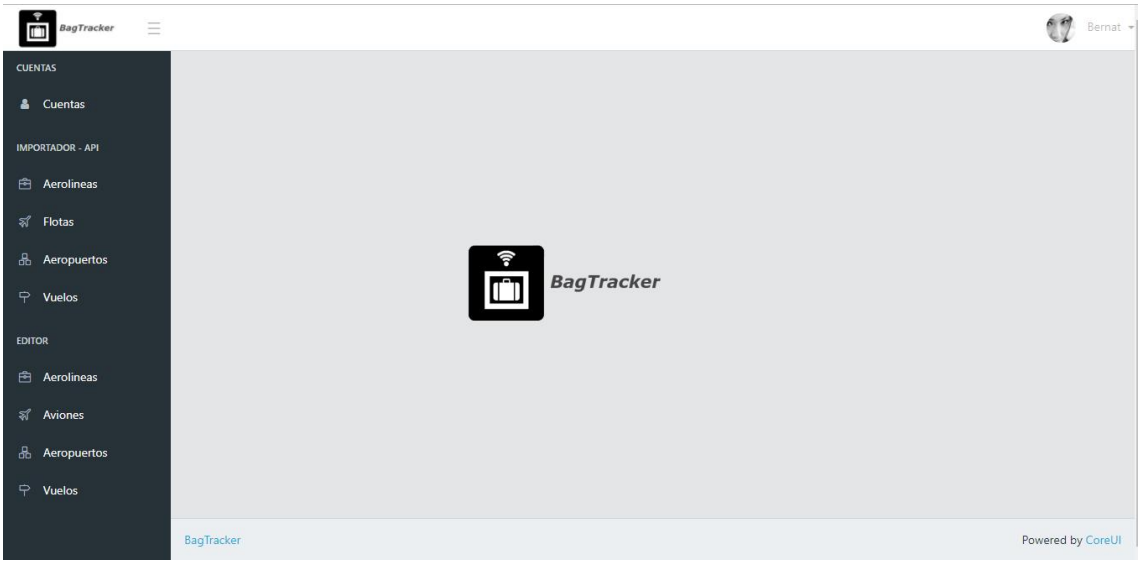


Fig. 2.2. Vista pàgina inicial

1.2.1.2. Comptes d'usuari

Vista comptes d'usuari

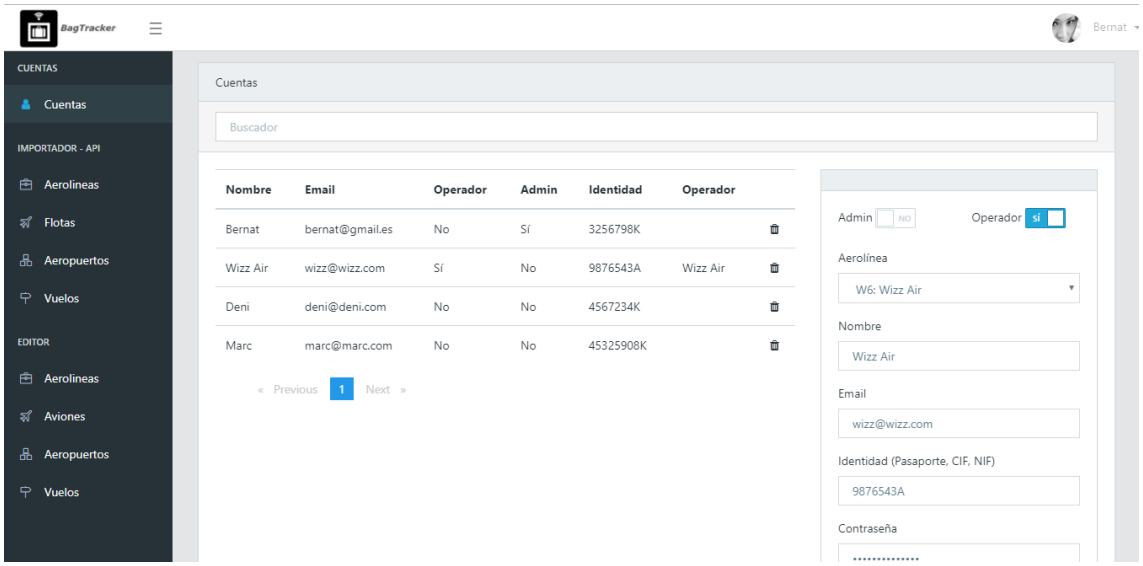


Fig. 2.3. Vista comptes d'usuari

1.2.1.3. Importadors – API externa

Vista importador d'aerolínies

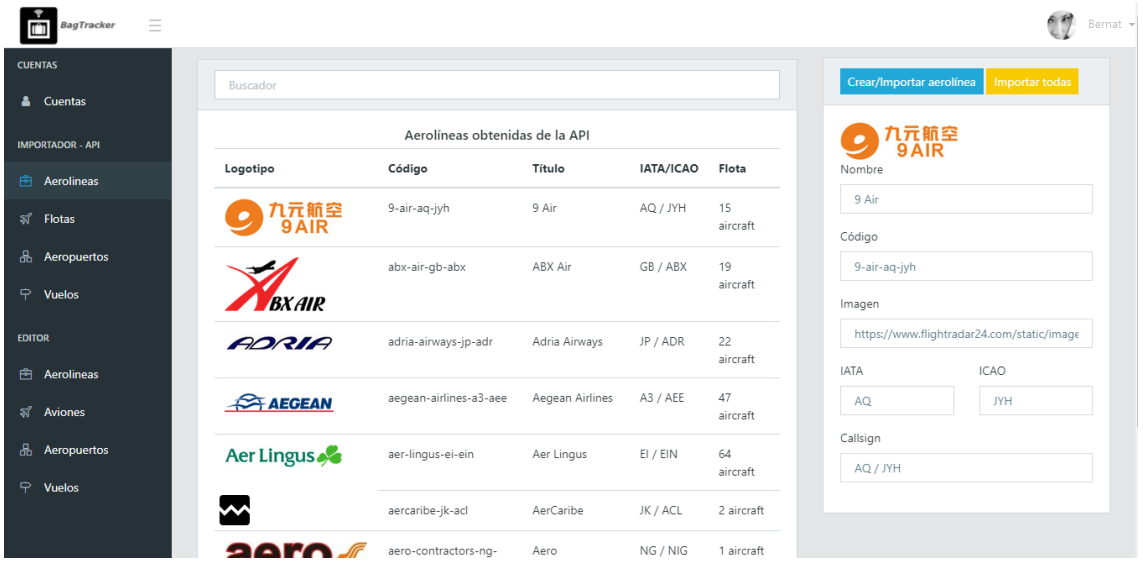


Fig. 2.4. Vista importador aerolínies

Vista importador de flotes d'avions

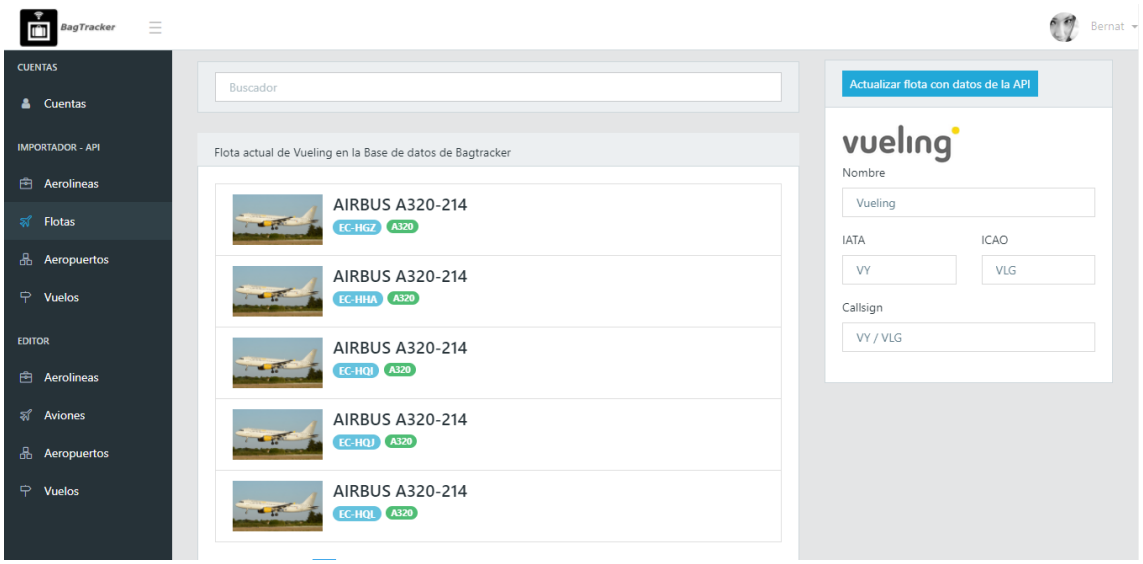


Fig. 2.5. Vista importador de flotes d'avions

Vista importador d'aeroports i països

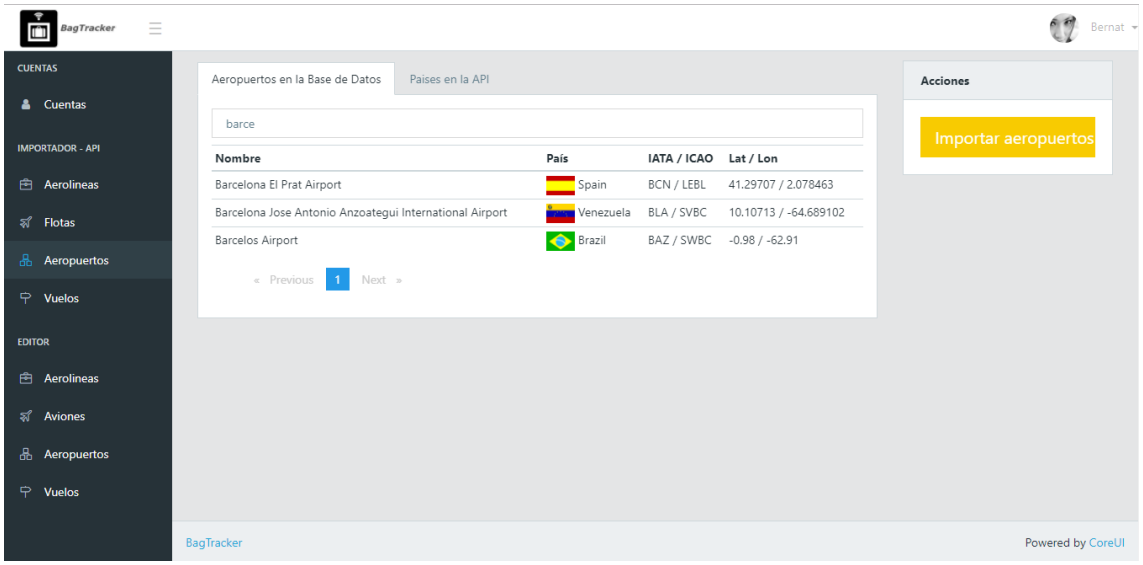


Fig. 2.6. Vista importador d'aeroports i països

Vista importador de vols

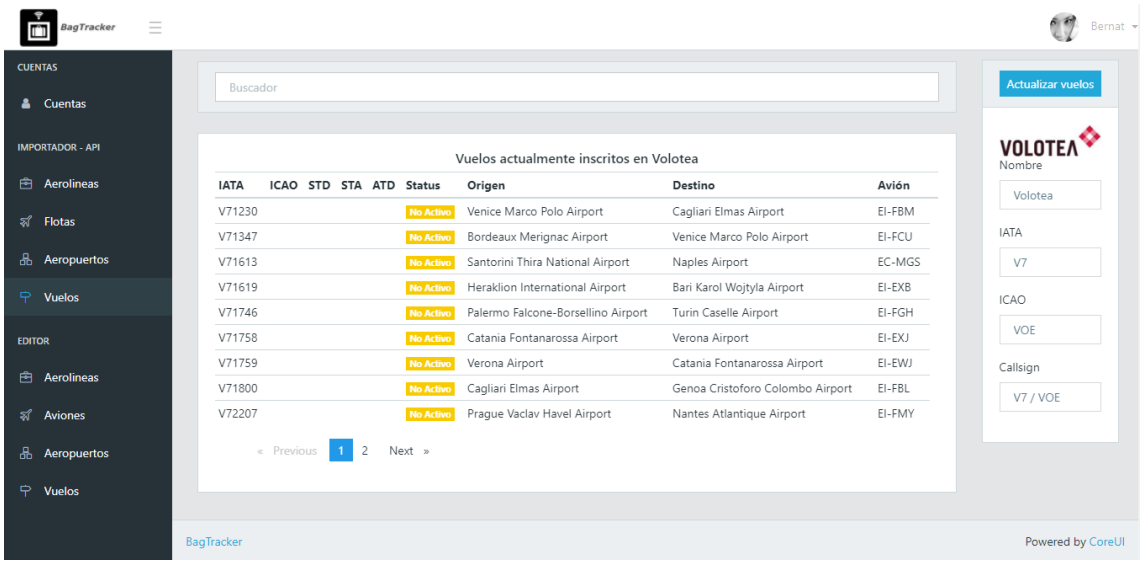


Fig. 2.8. Vista importador de vols

1.2.1.4. Editor

Vista editor d'aerolínies

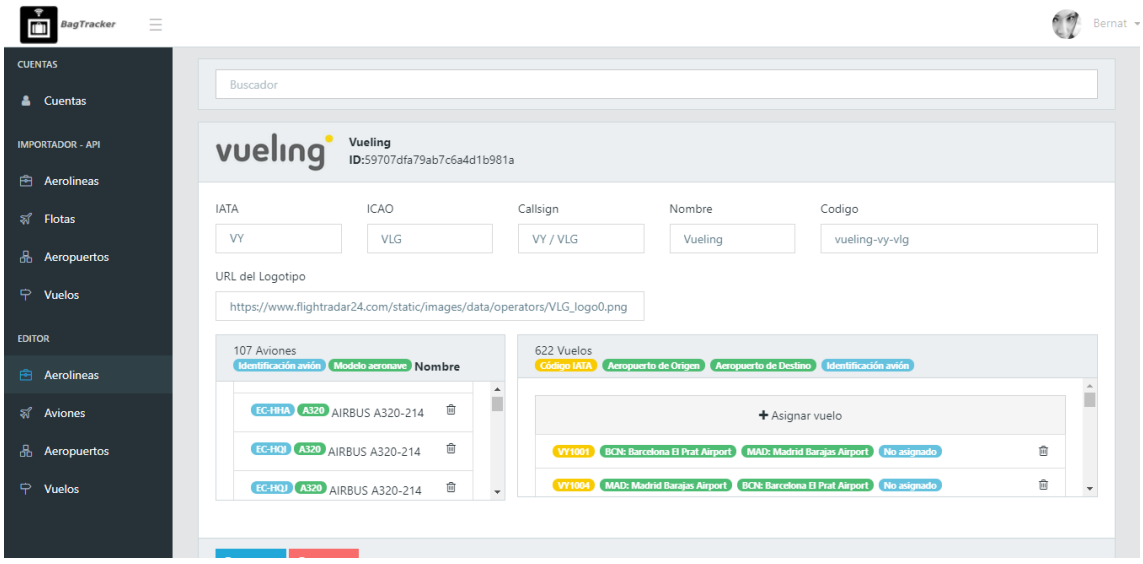


Fig. 2.9. Vista editor d'aerolínies

Vista editor d'avions

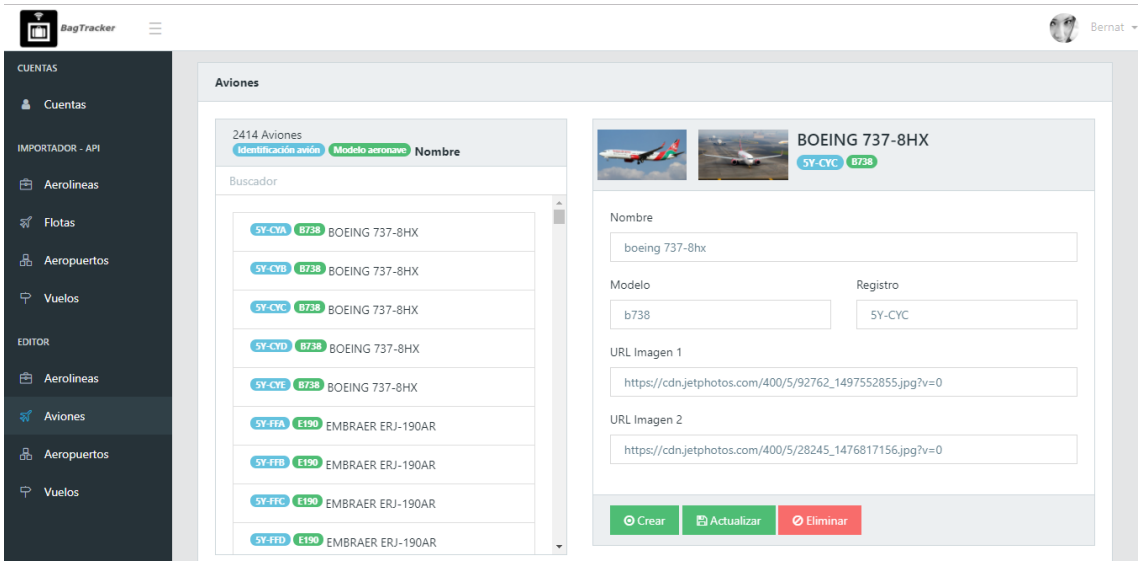


Fig. 2.10. Vista editor d'aerolínies

Vista editor d'aeroports

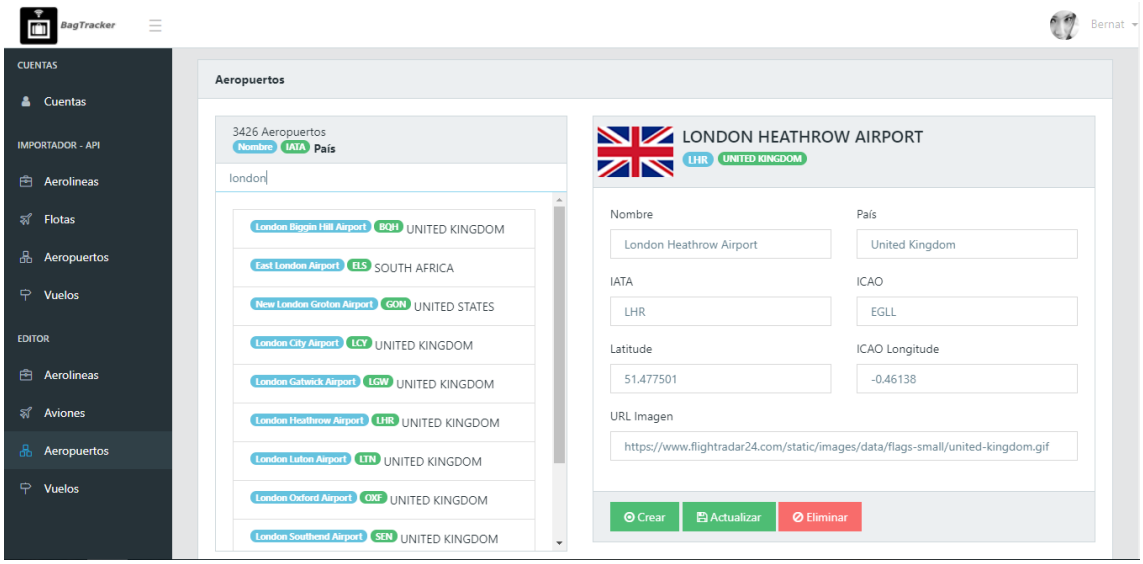


Fig. 2.11. Vista editor d'aeroports

Vista editor de vols

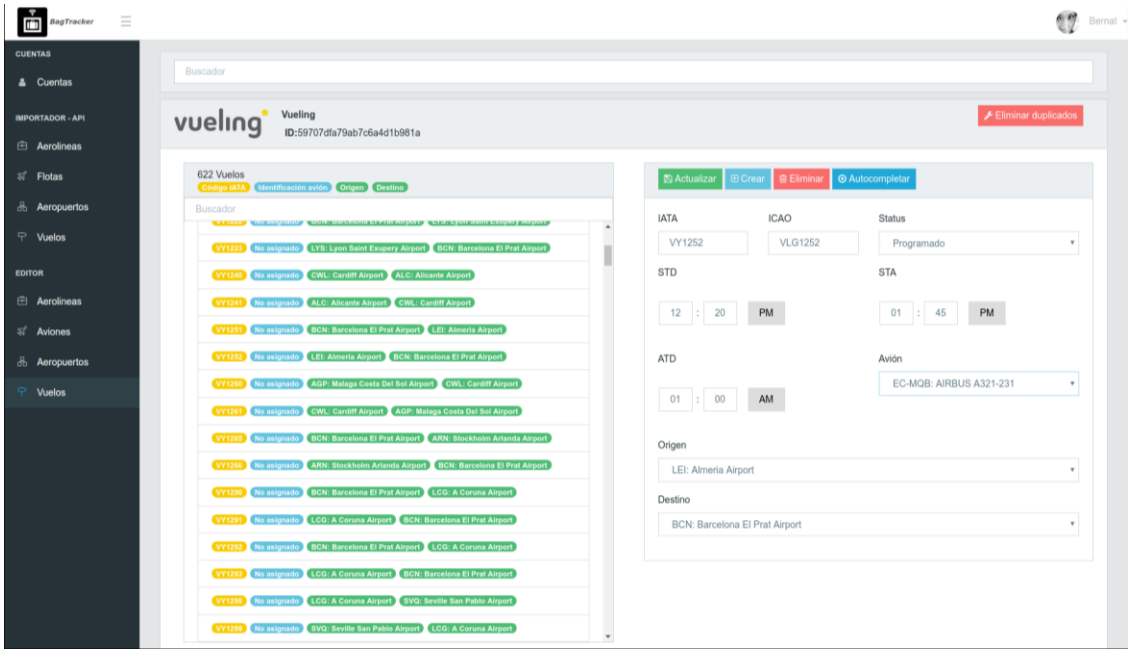


Fig. 2.12. Vista editor de vols

1.2.2. Operador

A continuació s'exposen les vistes de la plataforma web amb rol d'operador:

1.2.2.1. Vista de l'operador

Pàgina principal de BagTracker en la versió d'operador

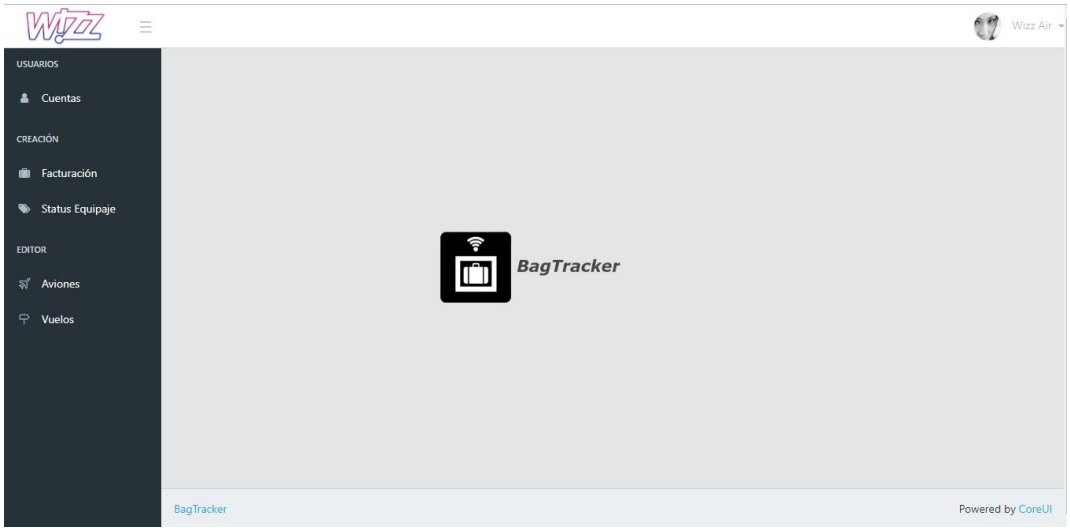


Fig. 2.13. Vista pàgina principal d'operador

1.2.2.2. Comptes d'usuari

Vista comptes d'usuari

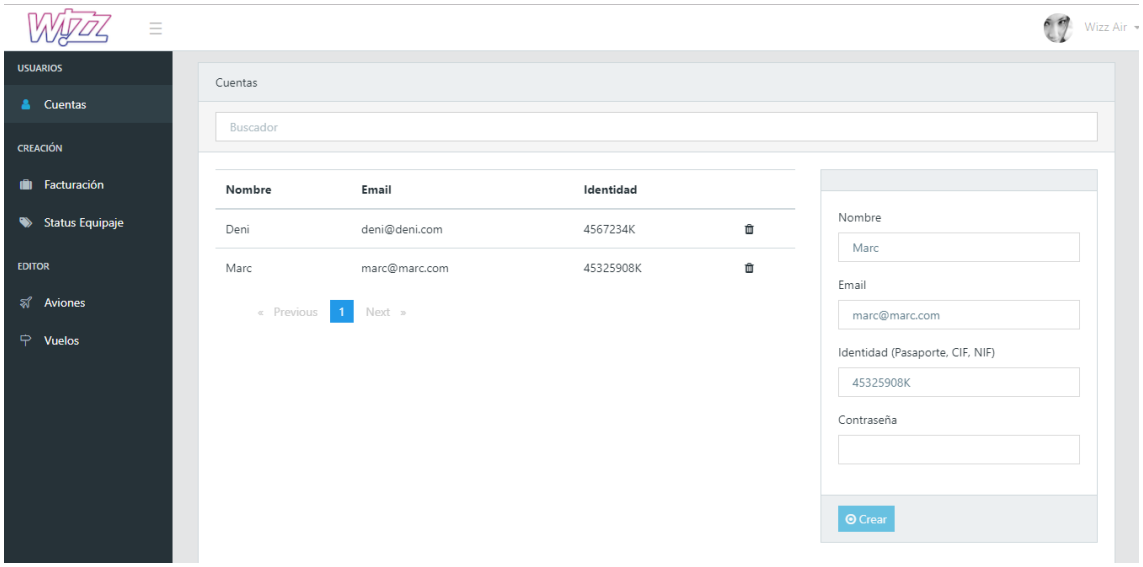


Fig. 2.14. Vista comptes d'usuari

1.2.2.3. Facturació i Status de l'equipatge

Vista de facturació

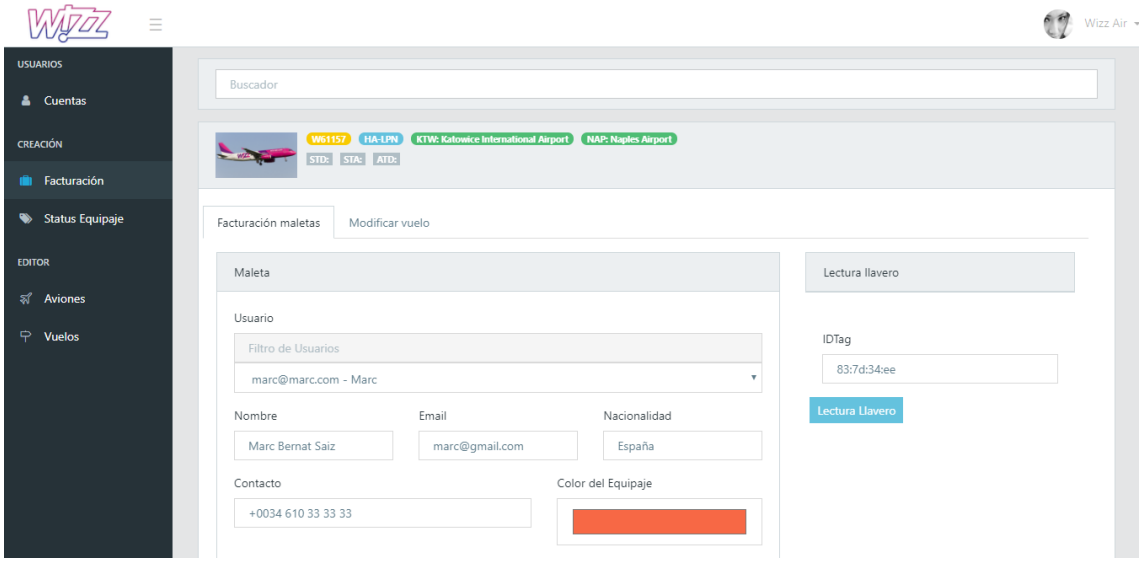


Fig. 2.15. Vista de facturació

Vista de simulació del estat de l'equipatge

En aquesta vista es vol simular la xarxa de sensors que registrant els esdeveniments segons per on passi l'equipatge:

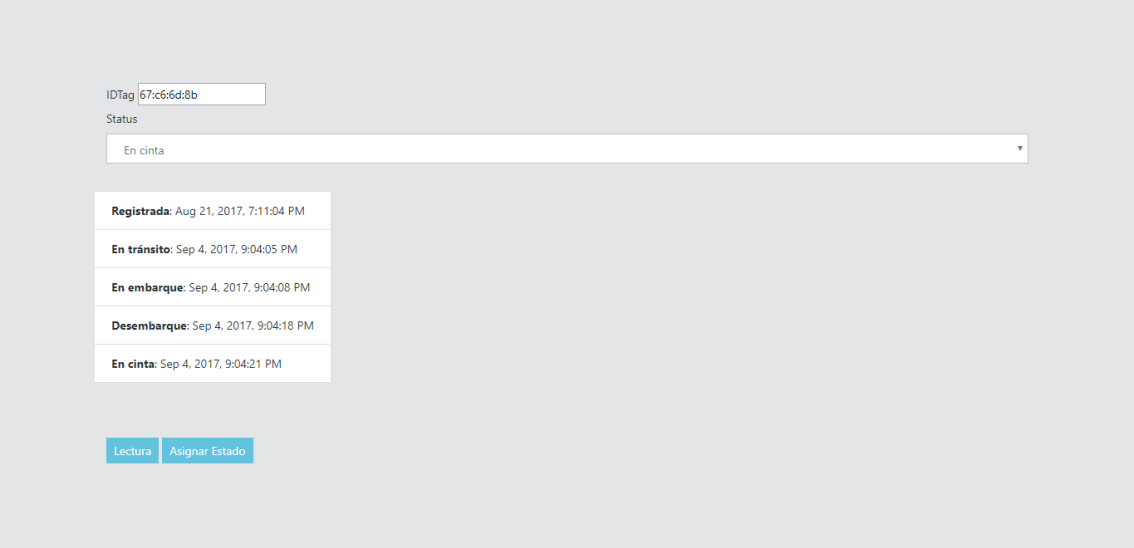


Fig. 2.16. Vista de simulació del estat de l'equipatge

1.2.2.4. Editor

Vista de l'editor d'avions de l'operador

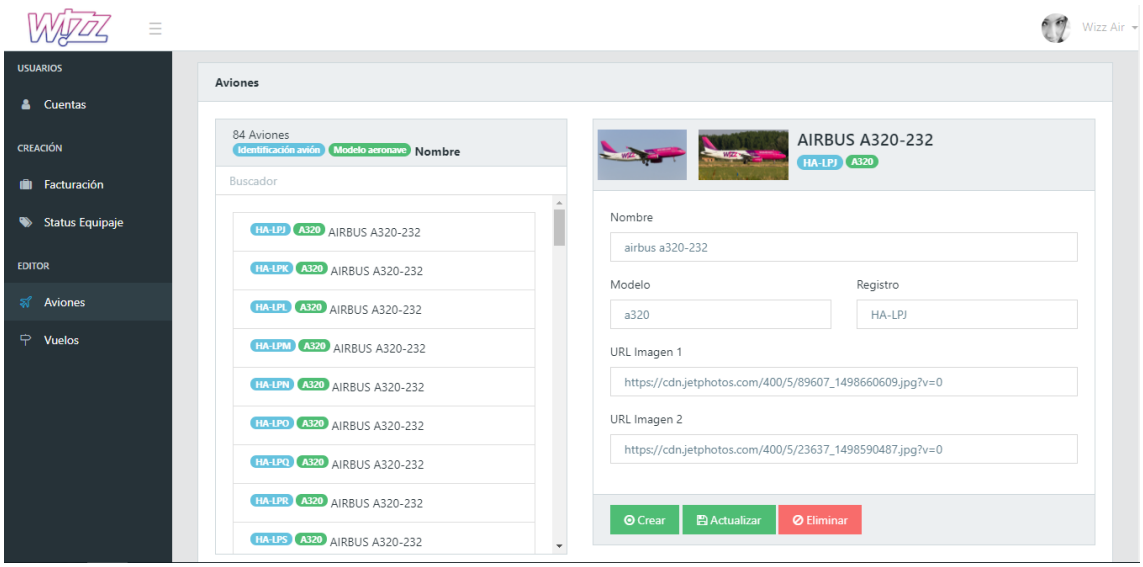


Fig. 2.17. Vista de l'editor d'avions de l'operador

Vista de l'editor de vols de l'operador

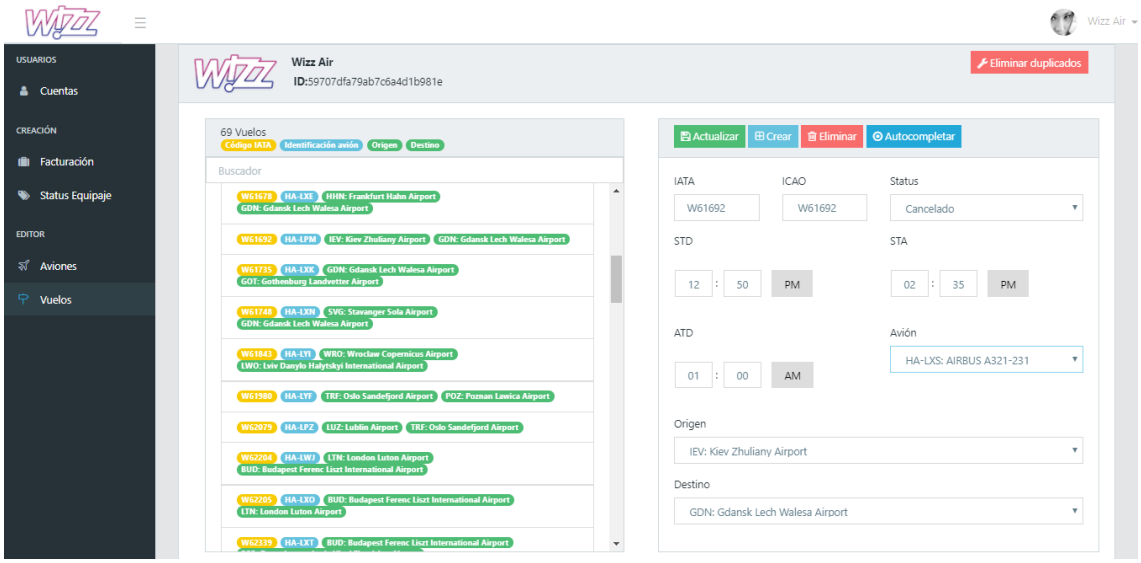


Fig. 2.17. Vista de l'editor de vols de l'operador

1.3. Disseny Aplicació Web

Inici de sessió per els usuaris finals i operadors:

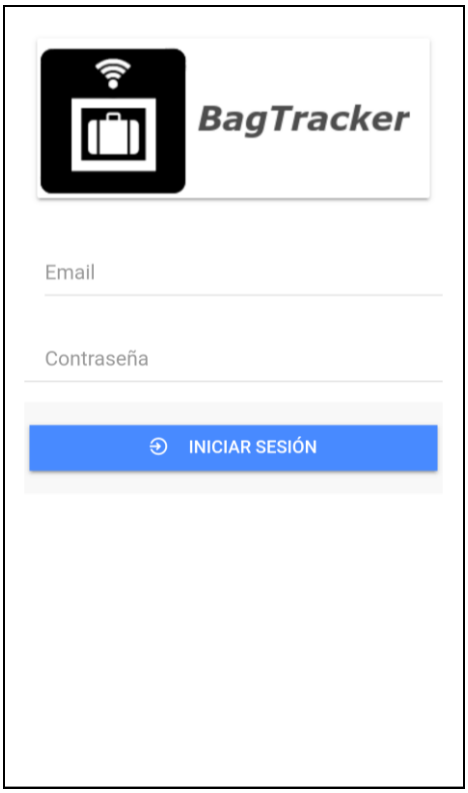


Fig. 3.1. Vista de l'inici de sessió

1.3.1. General

Vistes generals de l'aplicació mòbil

1.3.1.1. Lector

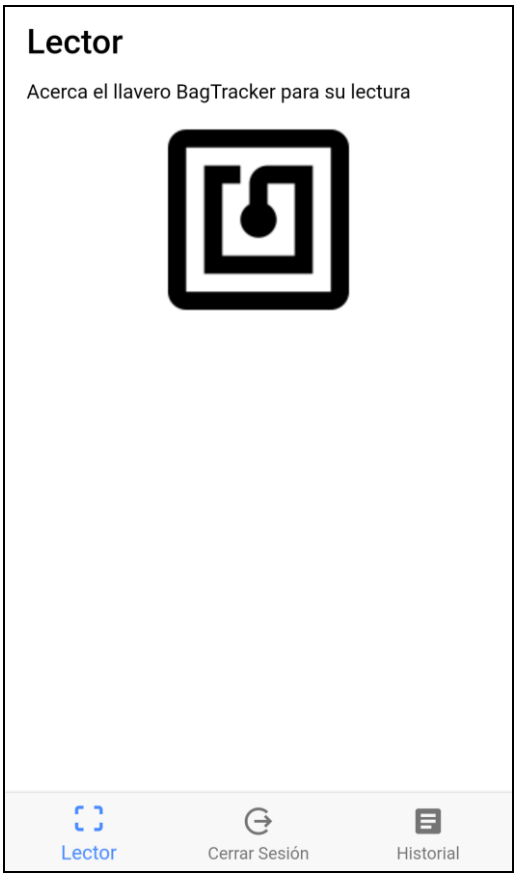


Fig. 3.2. Vista de lectura

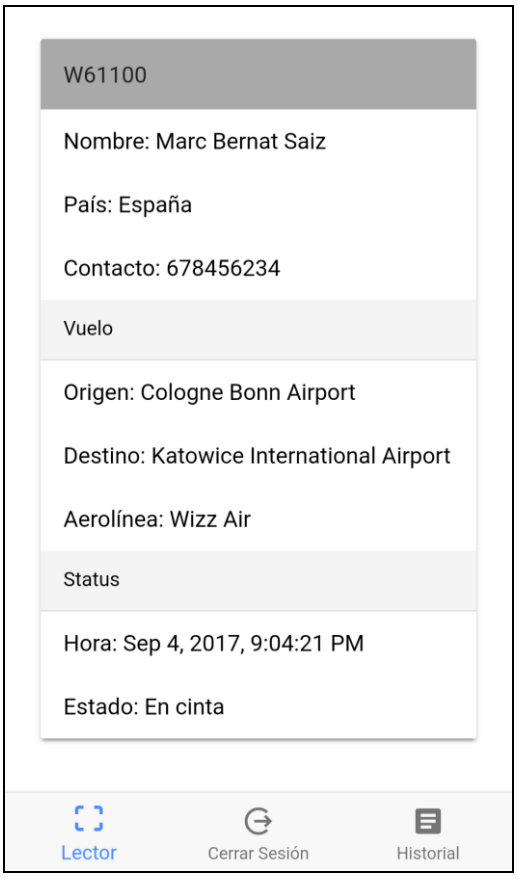


Fig. 3.3. Vista lectura realitzada

1.3.1.2. Historial

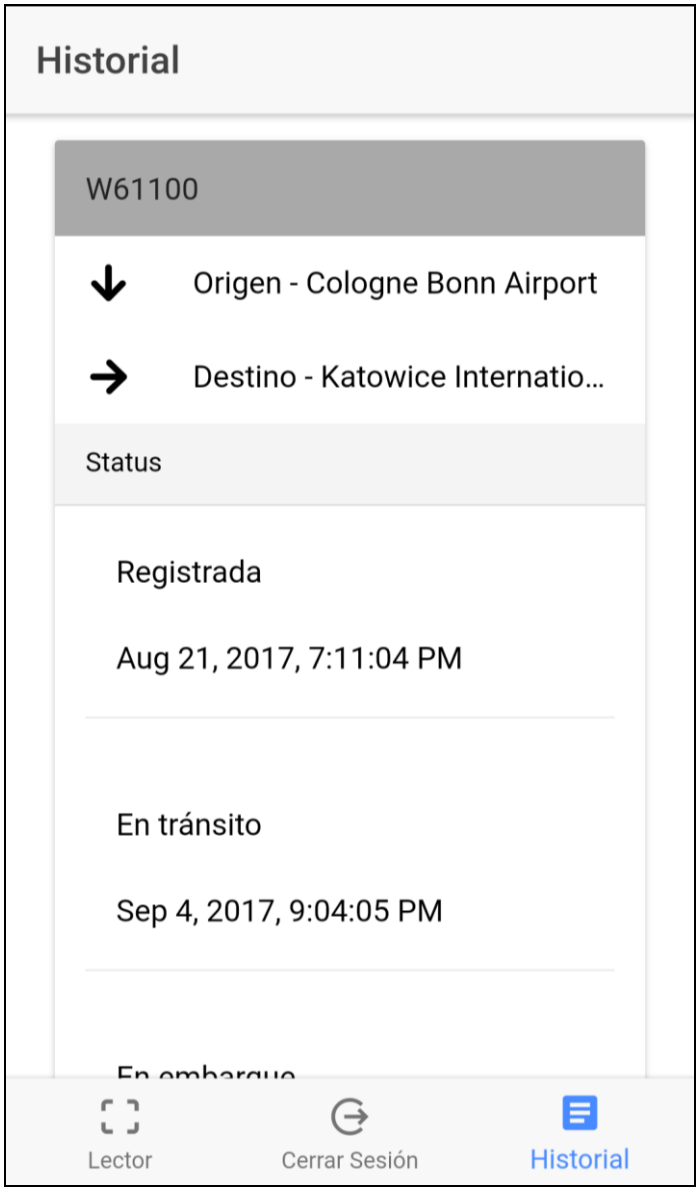


Fig. 3.4. Vista historial de l'usuari

1.4. GraphQL Schema Language

Fig. 4.1. GraphQL Schema Language

1.5. Datasheet Raspberry Pi

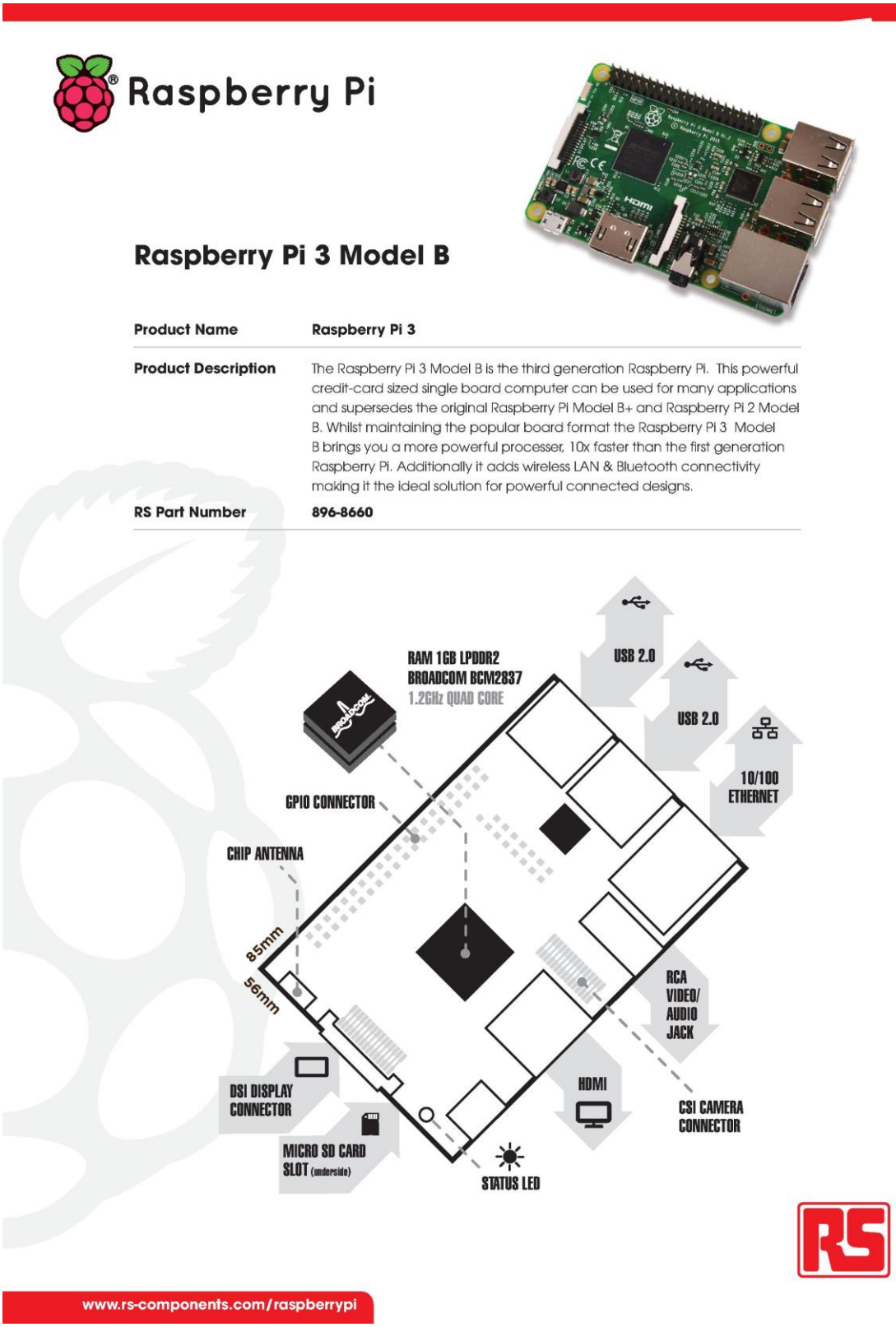


Fig. 5.1. Datasheet Raspberry Pi

Raspberry Pi

Raspberry Pi 3 Model B

Specifications

Processor	Broadcom BCM2387 chipset. 1.2GHz Quad-Core ARM Cortex-A53 802.11 b/g/n Wireless LAN and Bluetooth 4.1 (Bluetooth Classic and LE)
GPU	Dual Core VideoCore IV® Multimedia Co-Processor. Provides Open GL ES 2.0, hardware-accelerated OpenVG, and 1080p30 H.264 high-profile decode. Capable of 1Gpixel/s, 1.5Gtexel/s or 24GFLOPs with texture filtering and DMA infrastructure
Memory	1GB LPDDR2
Operating System	Boots from Micro SD card, running a version of the Linux operating system or Windows 10 IoT
Dimensions	85 x 56 x 17mm
Power	Micro USB socket 5V1, 2.5A

Connectors:

Ethernet	10/100 BaseT Ethernet socket
Video Output	HDMI (rev 1.3 & 1.4 Composite RCA (PAL and NTSC)
Audio Output	Audio Output 3.5mm jack, HDMI USB 4 x USB 2.0 Connector
GPIO Connector	40-pin 2.54 mm (100 mil) expansion header: 2x20 strip Providing 27 GPIO pins as well as +3.3 V, +5 V and GND supply lines
Camera Connector	15-pin MIPI Camera Serial Interface (CSI-2)
Display Connector	Display Serial Interface (DSI) 15 way flat flex cable connector with two data lanes and a clock lane
Memory Card Slot	Push/pull Micro SDIO

Key Benefits

- Low cost
- 10x faster processing
- Consistent board format
- Added connectivity

Key Applications

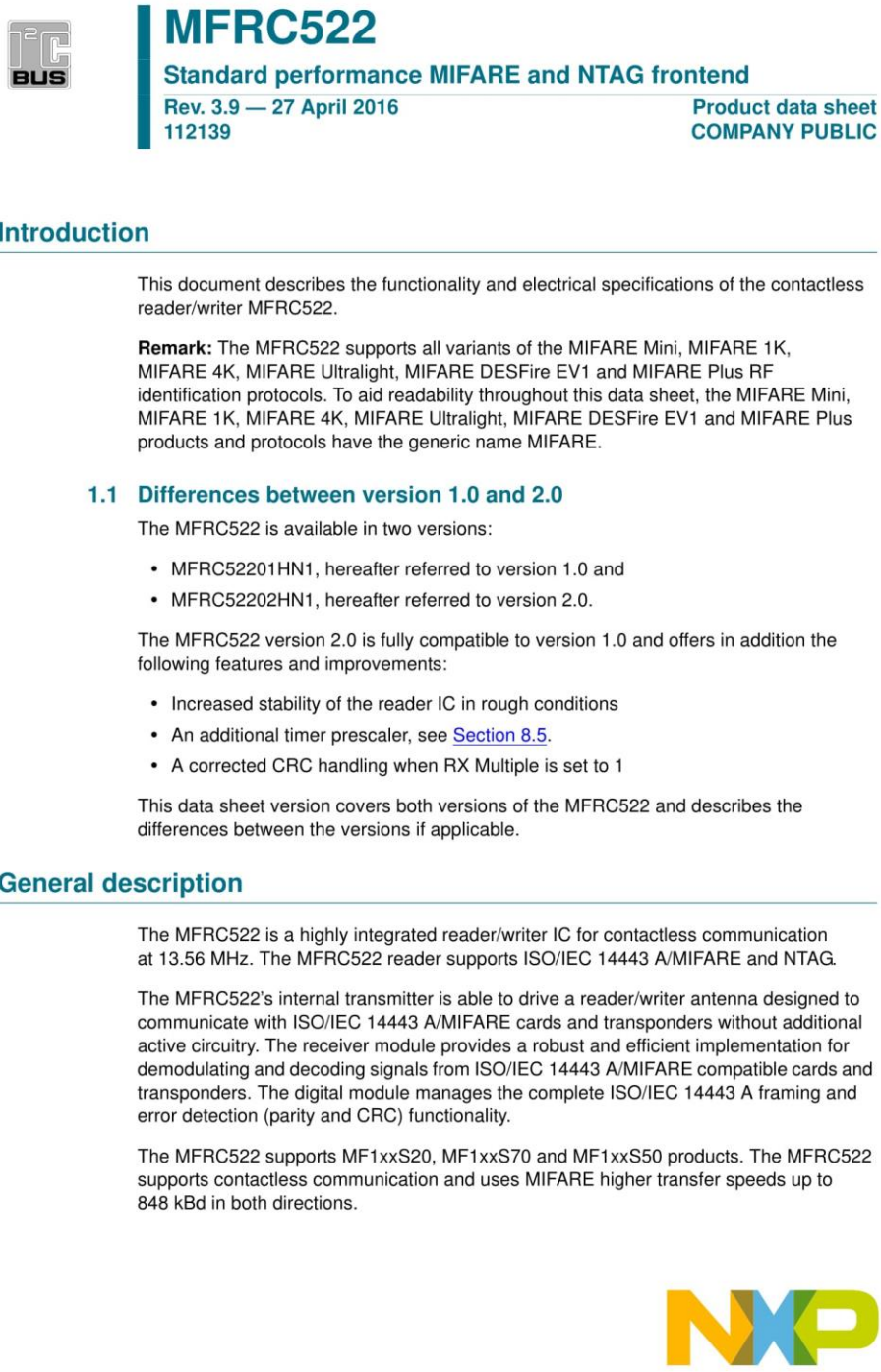
- Low cost PC/tablet/laptop
- Media centre
- Industrial/Home automation
- Print server
- Web camera
- Wireless access point
- IoT applications
- Robotics
- Server/cloud server
- Security monitoring
- Gaming
- Environmental sensing/monitoring (e.g. weather station)

RS

www.rs-components.com/raspberrypi

Fig. 5.2. Datasheet Raspberry Pi

1.6. Datasheet MFRC522



MFRC522

Standard performance MIFARE and NTAG frontend

Rev. 3.9 — 27 April 2016
112139

Product data sheet
COMPANY PUBLIC

1. Introduction

This document describes the functionality and electrical specifications of the contactless reader/writer MFRC522.

Remark: The MFRC522 supports all variants of the MIFARE Mini, MIFARE 1K, MIFARE 4K, MIFARE Ultralight, MIFARE DESFire EV1 and MIFARE Plus RF identification protocols. To aid readability throughout this data sheet, the MIFARE Mini, MIFARE 1K, MIFARE 4K, MIFARE Ultralight, MIFARE DESFire EV1 and MIFARE Plus products and protocols have the generic name MIFARE.

1.1 Differences between version 1.0 and 2.0

The MFRC522 is available in two versions:

- MFRC52201HN1, hereafter referred to version 1.0 and
- MFRC52202HN1, hereafter referred to version 2.0.

The MFRC522 version 2.0 is fully compatible to version 1.0 and offers in addition the following features and improvements:

- Increased stability of the reader IC in rough conditions
- An additional timer prescaler, see Section 8.5.
- A corrected CRC handling when RX Multiple is set to 1

This data sheet version covers both versions of the MFRC522 and describes the differences between the versions if applicable.

2. General description

The MFRC522 is a highly integrated reader/writer IC for contactless communication at 13.56 MHz. The MFRC522 reader supports ISO/IEC 14443 A/MIFARE and NTAG.

The MFRC522's internal transmitter is able to drive a reader/writer antenna designed to communicate with ISO/IEC 14443 A/MIFARE cards and transponders without additional active circuitry. The receiver module provides a robust and efficient implementation for demodulating and decoding signals from ISO/IEC 14443 A/MIFARE compatible cards and transponders. The digital module manages the complete ISO/IEC 14443 A framing and error detection (parity and CRC) functionality.

The MFRC522 supports MF1xxS20, MF1xxS70 and MF1xxS50 products. The MFRC522 supports contactless communication and uses MIFARE higher transfer speeds up to 848 kBd in both directions.

Fig. 6.1. Datasheet MFRC522

The following host interfaces are provided:

- Serial Peripheral Interface (SPI)
- Serial UART (similar to RS232 with voltage levels dependant on pin voltage supply)
- I²C-bus interface

3. Features and benefits

- Highly integrated analog circuitry to demodulate and decode responses
- Buffered output drivers for connecting an antenna with the minimum number of external components
- Supports ISO/IEC 14443 A/MIFARE and NTAG
- Typical operating distance in Read/Write mode up to 50 mm depending on the antenna size and tuning
- Supports MF1xxS20, MF1xxS70 and MF1xxS50 encryption in Read/Write mode
- Supports ISO/IEC 14443 A higher transfer speed communication up to 848 kBd
- Supports MFIN/MFOUT
- Additional internal power supply to the smart card IC connected via MFIN/MFOUT
- Supported host interfaces
  - SPI up to 10 Mbit/s
  - I²C-bus interface up to 400 kBd in Fast mode, up to 3400 kBd in High-speed mode
  - RS232 Serial UART up to 1228.8 kBd, with voltage levels dependant on pin voltage supply
- FIFO buffer handles 64 byte send and receive
- Flexible interrupt modes
- Hard reset with low power function
- Power-down by software mode
- Programmable timer
- Internal oscillator for connection to 27.12 MHz quartz crystal
- 2.5 V to 3.3 V power supply
- CRC coprocessor
- Programmable I/O pins
- Internal self-test

4. Quick reference data

Table 1. Quick reference data

Symbol	Parameter	Conditions		Min	Typ	Max	Unit
V_{DDA}	analog supply voltage	$V_{DD(PVDD)} \leq V_{DDA} = V_{DDD} = V_{DD(TVDD)}$; $V_{SSA} = V_{SSD} = V_{SS(PVSS)} = V_{SS(TVSS)} = 0$ V	[1][2]	2.5	3.3	3.6	V
V_{DDD}	digital supply voltage			2.5	3.3	3.6	V
$V_{DD(TVDD)}$	TVDD supply voltage			2.5	3.3	3.6	V
$V_{DD(PVDD)}$	PVDD supply voltage		[3]	1.6	1.8	3.6	V
$V_{DD(SVDD)}$	SVDD supply voltage	$V_{SSA} = V_{SSD} = V_{SS(PVSS)} = V_{SS(TVSS)} = 0$ V		1.6	-	3.6	V

Fig. 6.2. Datasheet MFRC522

1.7. Datasheet Clauer S50 Mifare 1K

MF1 IC S50

Functional specification

Rev. 5.2 — 15 January 2007 **Product data sheet**
001052 **PUBLIC**

1. General description

NXP has developed the Mifare MF1 IC S50 to be used in contactess smart cards according to ISO/IEC 14443A. The communication layer (Mifare RF Interface) complies to parts 2 and 3 of the ISO/IEC 14443A standard. The security layer sports the field proven CRYPTO1 stream cipher for secure data exchange of the Mifare Classic family.

1.1 Contactless Energy and Data Transfer

In the Mifare system, the MF1 IC S50 is connected to a coil with a few turns and then embedded in plastic to form the passive contactless smart card. No battery is needed. When the card is positioned in the proximity of the Read Write Device (RWD) antenna, the high speed RF communication interface allows to transmit data with 106 kBit/s.

1.2 Anticollision

An intelligent anticollision function allows to operate more than one card in the field simultaneously. The anticollision algorithm selects each card individually and ensures that the execution of a transaction with a selected card is performed correctly without data corruption resulting from other cards in the field.

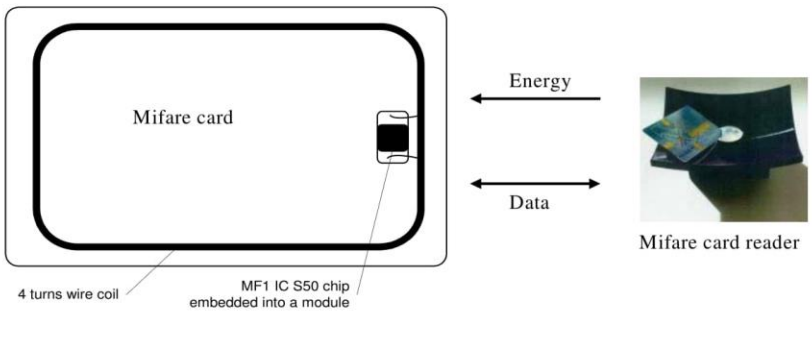


Fig 1. Mifare card reader

founded by Philips

Fig. 7.1. Datasheet Clauer RFID S50

1.3 User Convenience

The Mifare system is designed for optimal user convenience. The high data transmission rate for example allows complete ticketing transactions to be handled in less than 100 ms. Thus, the Mifare, card user is not forced to stop at the RWD antenna leading to a high throughput at gates and reduced boarding times onto busses. The Mifare card may also remain in the wallet during the transaction, even if there are coins in it.

1.4 Security

Special emphasis has been placed on security against fraud. Mutual challenge and response authentication, data ciphering and message authentication checks protect the system from any kind of tampering and thus make it attractive for ticketing applications. Serial numbers, which can not be altered, guarantee the uniqueness of each card.

1.5 Multi-application Functionality

The Mifare system offers real multi-application functionality comparable to the features of a processor card. Two different keys for each sector support systems using key hierarchies.

1.6 Delivery Options

- Die on wafer
- Bumped die on wafer
- Chip Card Module
- Flip Chip Package

2. Features

2.1 MIFARE, RF Interface (ISO/IEC 14443 A)

- Contactless transmission of data and supply energy (no battery needed)
- Operating distance: Up to 100mm (depending on antenna geometry)
- Operating frequency: 13.56 MHz
- Fast data transfer: 106 kbit/s
- High data integrity: 16 Bit CRC, parity, bit coding, bit counting
- True anticollision
- Typical ticketing transaction: < 100 ms (including backup management)

2.2 EEPROM

- 1 Kbyte, organized in 16 sectors with 4 blocks of 16 bytes each (one block consists of 16 byte)
- User definable access conditions for each memory block
- Data retention of 10 years.
- Write endurance 100.000 cycles

Fig. 7.2. Datasheet Clauer RFID S50